CENTRO UNIVERSITÁRIO UNIVATES
CENTRO DE CIÊNCIAS EXATAS E TECNOLÓGICAS
CURSO DE ENGENHARIA DA COMPUTAÇÃO

GERADOR AUTOMÁTICO DE EXERCÍCIOS PARA APOIO AO ENSINO DE PROGRAMAÇÃO

Pedro Ernesto Tramontina

Lajeado, novembro de 2015

Pedro Ernesto Tramontina

GERADOR AUTOMÁTICO DE EXERCÍCIOS PARA APOIO AO ENSINO DE PROGRAMAÇÃO

Trabalho de Conclusão de Curso apresentado ao Centro de Ciências Exatas e Tecnológicas do Centro Universitário UNIVATES, como parte dos requisitos para a obtenção do título de bacharel em Engenharia da Computação
Área de concentração: Algoritmos e Programação

ORIENTADOR: Marcelo de Gomensoro Malheiros

Lajeado, novembro de 2015

Pedro Ernesto Tramontina

GERADOR AUTOMÁTICO DE EXERCÍCIOS PARA APOIO AO ENSINO DE PROGRAMAÇÃO

Este trabalho foi julgado adequado para a obtenção do título de bacharel em Engenharia da Computação do CETEC e aprovado em sua forma final pelo Orientador e pela Banca Examinadora.

Orientador: _____

Prof. Marcelo de Gomensoro Malheiros, UNIVATES

Mestre pela UNICAMP – Campinas, Brasil

Banca Examinadora:

Prof. Evandro Franzen, UNIVATES

Mestre pela UFRGS – Porto Alegre, Brasil

Prof. Juliano Dertzbacher, UNIVATES

Mestre pela UFRGS – Porto Alegre, Brasil

Prof. Marcelo de Gomensoro Malheiros, UNIVATES

Mestre pela UNICAMP – Campinas, Brasil

Coordenador do Curso de Engenharia da Computação:_____

Prof. Marcelo de Gomensoro Malheiros

Lajeado, novembro de 2015.

Dedico este trabalho especialmente aos meus pais, meu irmão e minha namorada, além de todas as pessoas que me ajudaram e me apoiaram em todos os momentos difíceis.

AGRADECIMENTOS

Agradeço a todas as pessoas que me ajudaram de alguma forma a realizar este trabalho, sendo através da motivação ou do ensino da teoria e da pratica.

Agradeço especialmente aos meus pais, por todo o incentivo e apoio em todos os momentos da minha vida. Por terem me indicado os melhores caminhos e me fornecido toda a base para o meu crescimento pessoal e profissional.

Ao meu irmão pelos conselhos e pelo apoio, além dos momentos de diversão e amizade.

Também tenho um agradecimento especial à minha namorada, por ter me ajudado, escutado e motivado, além de ter me apoiado em todas as minhas decisões para alcançar meus sonhos e objetivos.

Aos professores, agradeço por toda a dedicação e ensinamentos, e especialmente ao meu orientador, que me ajudou muito na realização deste trabalho.

Também agradeço aos colegas do curso pelo auxílio nas tarefas desenvolvidas durante toda a trajetória percorrida nestes longos anos de faculdade.

RESUMO

Através de pesquisas realizadas na área de ensino, especificamente na área de programação, foi identificado que o método utilizado para ensinar a programar ainda segue o formato tradicional, limitando o processo de aprendizado e causando uma grande taxa de reprovação em disciplinas como as de algoritmos e programação. Com isso, diversas ferramentas estão sendo desenvolvidas para melhorar o processo de ensino e aumentar a disseminação do conhecimento. Além disso, técnicas como a gamificação estão sendo aplicadas para aumentar a motivação dos alunos na busca pelo conhecimento. Entretanto, percebeu-se que estes ambientes *online* de apoio ao ensino que disponibilizam exercícios, possuem apenas uma lista simplificada de atividades, obrigando os usuários a realizarem sempre as mesmas tarefas. Desta forma, este trabalho tem como objetivo desenvolver um software gerador de exercícios de programação, visando uma variação das atividades nestas ferramentas de autoestudo, além de auxiliar professores no processo de ensino, facilitando a criação e disponibilização de listas de exercícios.

Palavras-chave: Ensino de Programação, Gerador de Exercícios, Ferramentas de Autoestudo, Pseudocódigo.

ABSTRACT

Through researches realized on teaching area, specifically on programming field, it was identified that the programming teaching methodology still follows the traditional format, limiting the learning process and causing a large failure rate in disciplines as the programming ones. Therewith, several tools have been developed to improve the teaching process and increase the knowledge dissemination. Besides, techniques as gamification have been used to increase student motivation in their search for knowledge. However, it was realized that those teaching online support environments which provide exercises, contain only a simple activity list, forcing users to always make the same tasks. Therefore, this paper has the purpose of developing a programming exercise generator, aiming a variation for the activities on those self study tools, besides helping teachers with the teaching process, making easier creating and providing exercise lists.

Keywords: Programming Teaching, Exercise Generator, Self Study Tools, Pseudocode.

LISTA DE FIGURAS

LISTA DE CÓDIGOS

LISTA DE TABELAS

LISTA DE ABREVIATURAS

JAR:	Java Archive
JRE:	Java Runtime Environment
MOOC:	Massive Open Online Course
MIT:	Massachusetts Institute of Technology
RP:	Redeemable Points – Pontos Resgatáveis
VPL:	Virtual Programming Lab
XP:	Experience Points – Pontos de Experiência

SUMÁRIO

1 INTRODUÇÃO

Com o avanço da tecnologia e o uso da Internet, diferentes formas de ensinar e aprender começam a surgir, possibilitando a realização de cursos a distância e o desenvolvimento de ambientes *online* de autoestudo. Dentre estes novos modelos de estudo, está o Massive Open Online Course (MOOC), que é uma forma de ensino a distância voltada para um grande número de pessoas, sendo realizado através da internet.

Vale ressaltar que a motivação é um elemento importante para este novo método de ensino. Matta e Figueiredo (2013) comentam que a taxa de desistência em cursos MOOC é muito elevada. Desta forma, novas técnicas motivacionais estão sendo pesquisadas para incentivar alunos na busca pelo conhecimento, sendo a gamificação uma delas.

A área de Tecnologia da Informação também está iniciando um processo de melhorias nas formas de ensinar, sendo o ensino de programação um assunto que está recebendo grande atenção. Atualmente, o método utilizado para ensinar a programar ainda segue o formato tradicional, trazendo limitações ao processo de aprendizado e causando também uma grande taxa de reprovação (JÚNIOR; RAPKIEWICZ, 2004). Com isso, ferramentas *online* que visam ensino de programação e aplicam técnicas de gamificação já estão sendo desenvolvidas.

Através de pesquisas e análises sobre estas ferramentas, foi identificado que vários destes ambientes disponibilizam uma lista simples ou uma sequência estática de atividades, forçando os usuários a realizarem sempre os mesmos exercícios. Desta forma, um sistema que possa gerar automaticamente exercícios de programação, não apenas com textos preestabelecidos, mas com possibilidade de variação dos elementos contidos na atividade, seria bastante útil para a prática das atividades que envolvem o ensino de algoritmos e programação.

Este trabalho tem como objetivo o desenvolvimento de um sistema capaz de gerar exercícios de programação automaticamente, gerando enunciados de atividades juntamente com suas soluções em diversas linguagens de programação. O sistema proposto poderá ser

utilizado tanto nas ferramentas de autoestudo, quanto diretamente por professores das disciplinas de algoritmos e programação.

Especificamente, as ações principais do trabalho serão: desenvolver o sistema gerador de exercícios, criar um pseudocódigo robusto que servirá como base para gerar a solução dos enunciados, desenvolver dentro do sistema uma forma de exibir dicas para os alunos a partir do pseudocódigo, documentar e publicar o código-fonte.

Cabe ressaltar que este sistema será desenvolvido visando uma melhora no processo de ensino de programação, pois além de gerar variações de exercícios em um ambiente *online* para autoestudo, tal sistema será de grande utilidade para professores que almejam diversificar listas de exercícios ou atividades para serem aplicadas em testes.

Além disso, o uso do pseudocódigo e das dicas geradas pelo sistema podem ser utilizado pelos alunos para auxiliar na solução dos exercícios, melhorando o entendimento da lógica de programação.

Estre trabalho está organizado da seguinte forma: no segundo capítulo é apresentada toda a pesquisa feita e o referencial teórico utilizado. O Capítulo 3 descreve a etapa de desenvolvimento, onde foi realizado a implementação do sistema proposto. O Capítulo 4 demonstra os resultados obtidos, detalhando o funcionamento e os casos de uso. Por fim, o Capítulo 5 descreve a conclusão e considerações finais do trabalho.

2 REVISÃO DE LITERATURA

Este capítulo tem por finalidade dar uma visão geral das pesquisas recentes na área de educação e ensino de programação, bem como apresentar algumas ferramentas existentes que têm o intuito de apoiar o aprendizado de lógica e linguagens de programação. Este levantamento foi realizado visando estabelecer o que já existe e apontar para as novas demandas nesta área da Tecnologia da Informação.

Ferramentas que buscam melhorar a qualidade do ensino e disseminar a educação além de ambientes e técnicas para melhorar especificamente o ensino de programação, serão abordados nas seções a seguir.

2.1 Massive Open Online Course

Com a popularização dos computadores e o uso intenso da Internet, novas formas de estudo e de aprendizado começaram a surgir, possibilitando autonomia na busca pelo conhecimento. Desta forma, diversos modelos de cursos e várias plataformas focadas em autoestudo começaram a ser desenvolvidas. Estes modelos criam uma interação simultânea entre milhares de pessoas, além de serem acessíveis de qualquer lugar. Um caso particular destas plataformas é conhecida como Massive Open Online Course (MOOC).

Os MOOCs têm recebido recentemente uma grande atenção por parte da mídia e dos profissionais da educação em virtude da promessa de prover acesso livre a cursos de qualidade, com potencial de baixar os custos da educação e até mesmo modificar os atuais modelos de ensino (YUAN; POWELL, 2013).

Segundo Matta e Figueiredo (2013, p. 2) os MOOCs “oferecem uma oportunidade estratégica para melhorar a qualidade da educação, bem como facilitar o diálogo político, a partilha de conhecimento e a capacitação das pessoas ao redor do mundo”.

2.1.1 Histórico do MOOC

Em relação ao histórico do MOOC, é importante destacar que foi a partir dos anos 2000 que formulações mais concretas sobre educação aberta começaram a surgir. Em 2002, por exemplo, o Massachusetts Institute of Technology (MIT) lançou o projeto OpenCourseWare, com 50 cursos abertos na Internet, e com o objetivo de promover o conhecimento e educar estudantes. No ano de 2007 surgiram outras propostas de cursos de forma aberta e massiva, onde o primeiro que foi ofertado ocorreu na Utah State University (ALBERTI et al., 2013).

Porém, o termo Massive Open Online Course surgiu somente em 2008 para descrever o curso de Conectivismo e Conhecimento Conectivo, que foi ministrado por George Siemens e Stephen Downes (BARIN; BASTOS, 2013). Este curso *online* foi inicialmente designado para um grupo de 25 estudantes da Universidade de Manitoba, no Canadá, e aberto para todas as pessoas do mundo, que estivessem previamente registradas e tivessem interesse em aprender sobre aquele assunto. Como resultado, mais de 2.300 pessoas participaram do curso remotamente, sem pagar qualquer valor ou taxa de inscrição (BARIN; BASTOS, 2013; YUAN; POWELL, 2013).

Em 2011, por uma iniciativa de Sebastian Thrun e alguns colegas da Universidade de Stanford, foi liberado acesso para o curso de Introdução à Inteligência Artificial que estavam realizando. Esta ação atraiu mais de 160.000 estudantes de mais de 190 países, e o fato repercutiu mundialmente. Após este evento, os próprios professores desta universidade fundaram a instituição chamada Coursera (ALBERTI, et al., 2013; YUAN; POWELL, 2013). Segundo Alberti et al. (2013, p. 4) “a Coursera, numa parceria com várias universidades, oferece dezenas de cursos em várias áreas e conta com mais de um milhão de inscritos procedentes de mais de 196 países”.

O jornal The New York Times proclamou que 2012 seria o ano do MOOC, pois segundo o próprio jornal, em setembro daquele mesmo ano, os provedores dos cursos MOOC possuíam um grande número de estudantes. Um dos grandes provedores destes cursos, conhecido como EDX tinha 370.000 alunos e o Udacity possuía em torno de 150.000 estudantes. Já o Coursera, somava mais de 1,7 milhões de estudantes inscritos (MATTA; FIGUEIREDO, 2013).

Nos últimos anos, os MOOCs têm se disseminado através de várias plataformas espalhadas pelo mundo, e o Brasil vem acompanhando este ritmo, já que é o segundo país em número de inscrições em cursos gratuitos de forma *online* disponibilizados por universidades

estrangeiras através do Coursera. Dentre todas as inscrições, que ficam em torno de duas milhões, 5,9% são de alunos brasileiros (BARIN; BASTOS, 2013).

Segundo Yuan e Powell (2013), o modelo de ensino MOOC tem crescido e se tornou uma referência para iniciativas de cursos *online* de várias instituições de ensino e até mesmo organizações comerciais.

2.1.2 Visão Geral de um MOOC

De uma maneira geral, o MOOC é um modelo de curso, realizado de forma *online* e oferecido livremente para todas as pessoas de qualquer lugar, através da Internet (MATTA; FIGUEIREDO, 2013). Conforme descrito por Matta e Figueiredo (2013, p. 2), este tipo de curso "possui como característica principal o fato de permitir um engajamento ativo de dezenas ou centenas de milhares de estudantes que auto-organizam sua participação de acordo com suas metas, conhecimentos prévios, habilidades e interesse comum".

Duas características merecem destaque no conceito do MOOC: o acesso aberto e a escalabilidade. Quanto ao acesso aberto, todos que desejam participar não precisam estar regularmente matriculados e nem pagar qualquer valor ou taxa de inscrição. Referente à escalabilidade, pela proposta de um MOOC, o curso deve ser projetado para ter um número indefinido de participantes, e suportar um crescimento exponencial de inscrições, além de poder aumentar o número de vagas inicialmente proposto dependendo da demanda do curso (BARIN; BASTOS, 2013; BASTOS; BIAGIOTTI, 2014).

Segundo Matta e Figueiredo (2013), existe uma integração de três elementos dentro do modelo MOOC: conhecimento de especialistas em áreas específicas, recursos *online* abertos, e a conectividade das redes sociais. Neste mesmo sentido, Bastos e Biagiotti (2014) destacam que com o uso das redes sociais e outras ferramentas de integração e participação, existe uma expansão do conhecimento por parte de todos os envolvidos, gerando um conhecimento que vai além do conteúdo específico do curso. "Conhecimento gerando mais conhecimento, quebrando paradigmas e criando uma poderosa semente para romper com a clássica forma de ensinar e aprender" (BASTOS; BIAGIOTTI, 2014, p. 3).

Bastos e Biagiotti (2014) citam ainda uma outra característica que os MOOC têm apresentando: a capacidade de ser utilizado como um material de apoio nas aulas presenciais. Segundo os autores, muitos desses cursos podem ajudar os professores em sala de aula, em virtude de apresentarem bons materiais e conteúdo de ótima qualidade.

Entretanto, o fato de um MOOC ser um curso aberto, não tira a possibilidade de um estudante ter que possuir um conhecimento prévio, ou seja, em alguns casos, pode haver a necessidade de que exista um conhecimento mínimo para participação de determinados cursos (MATTA; FIGUEIREDO, 2013).

Matta e Figueiredo relatam que um outro aspecto sobre o MOOC deve ser comentado. Apesar da participação nos cursos desta modalidade serem oferecidos gratuitamente, para que um aluno concluinte possa obter o certificado de participação, é necessário que exista o pagamento de um determinado valor. “Assim, universidades e os provedores de MOOC estão contando que os concluintes irão criar a principal fonte de receita para estas instituições” (MATTA; FIGUEIREDO, 2013, p. 6).

Mesmo que para obtenção do certificado ou de créditos universitários, o pagamento é necessário, o acesso aos cursos e ao conteúdo é disponibilizado de forma gratuita, e assim, programas baseados em MOOC democratizam o acesso à educação (BARIN; BASTOS, 2013). Sua evolução está em fase de crescimento, mas sem uma forma definida, e por isso, há muito o que se pesquisar neste método de ensino (BASTOS; BIAGIOTTI, 2014; MATTA; FIGUEIREDO, 2013). Mas pode-se afirmar que com este modelo de educação “o conhecimento rompeu as barreiras das universidades e está disponível para aqueles que se habilitam a experimentar” (MATTA, FIGUEIREDO, 2013, p. 12).

2.1.3 Categorias do MOOC

Referente à categorização, os MOOCs dividem-se em dois grupos: o xMOOC e o cMOOC. Estes dois formatos possuem características distintas, pois a forma de ensino e estudo são realizadas diferentemente em cada modelo (ALBERTI, et al., 2013).

O modelo xMOOC tem uma maior semelhança às aulas tradicionais de cursos presenciais, onde é o professor que exerce a função de mediador das aulas além de ser o disseminador do conhecimento (BASTOS; BIAGIOTTI, 2014). O planejamento e a produção dos materiais didáticos são realizados previamente pelo especialista da área, e desta forma, as interações e colaborações dos alunos ocorre focada no conteúdo disponibilizado pelo professor (ALBERTI, et al., 2013; BASTOS; BIAGIOTTI, 2014).

Sobre o xMOOC, Bastos e Biagiotti (2014, p. 3) afirmam que:

> O criador do conteúdo ministrado é o professor (geralmente são utilizadas vídeo aulas pré-gravadas). O caminho a ser percorrido pelo aluno é guiado também pelo

> professor, com exercício de fixação em etapas gradativas. O debate entre os alunos é incentivado, porém, é direcionado pelo tutor.

Segundo Alberti et al. (2013), além dos conteúdos, as avaliações e trabalhos também estão centradas nos materiais didáticos do instrutor.

Já o modelo cMOOC é baseado no conectivismo, sendo este um método diferente de ensino a distância, onde o aprendizado é embasado no conceito de rede e os próprios alunos ajudam com o conteúdo do curso (ALBERTI et al., 2013). Este modelo enfatiza a conexão e o ensino colaborativo. O princípio desta metodologia está no conhecimento da comunidade, e o conteúdo apresentado pelos professores e orientadores auxiliam em uma fase inicial. Sequencialmente, haverá uma pesquisa de cada estudante e, então, o compartilhamento entre os participantes (ALBERTI et al., 2013; BASTOS; BIAGIOTTI, 2014; YUAN; POWELL, 2013).

Bastos e Biagiotti (2014, p. 3) comentam que:

> Os participantes são incentivados a disponibilizar conteúdos externos que venham a enriquecer o debate, por meio de *blogs* e redes sociais. Pessoas interessadas sobre um mesmo tema aprofundam o debate e o professor está no mesmo patamar hierárquico dos alunos, contribuindo e orientando as discussões. O conteúdo é construído colaborativamente pela comunidade de aprendizado.

Apesar de explorar novas formas de pedagogia além das tradicionais formas de ensino em sala de aula, este modelo de ensino ainda não é bem aceito nas academias e instituições de ensino, e tem sido alvo de várias críticas (BASTOS; BIAGIOTTI, 2014; YUAN; POWELL, 2013).

2.1.4 Desafios e dificuldades com o uso do MOOC

Apesar da potencialidade provida pelos MOOC em ofertar ensino de qualidade e com baixo custo, algumas dificuldades e desafios podem ser comentados.

Barin e Bastos (2013) citam que um desafio será a forma de certificar e validar os cursos oferecidos, pois isso implica diretamente no número de pessoas interessadas. Outro desafio comentado pelos autores é a respeito da adaptação dos novos papéis dos professores em ambientes de aprendizagem abertos. Alberti et al. (2013) comenta ainda, que devido à ausência de processos seletivos e o pressuposto de que todos os participantes possuem

fluência no conteúdo abordado, existe uma falta de credibilidade que causa uma limitação referente ao interesse de participação das pessoas.

Outro aspecto que pode ser considerado como uma dificuldade, e consequentemente um desafio, é que grande parte das universidades não devem dar créditos formais aos estudantes que concluem estes cursos, já que normalmente estas instituições tradicionais não reconhecem o valor de certificados MOOC (BASTOS; BIAGIOTTI, 2014).

Porém, uma das maiores dificuldades encontradas nos MOOC é a alta taxa de desistência de alunos. Matta e Figueiredo (2013) citaram o curso Introduction to Astronomy do provedor Coursera. Neste caso, 60.000 estudantes se inscreveram para participar, mas somente 3,5% deles terminaram. No curso Introduction to Solid State Chemistry, do EDX, dos 28.500 alunos inscritos, somente 1,7% finalizaram todas as aulas.

Neste caso, para êxito nos estudos a distância, a autodisciplina e a motivação são essenciais. Para tentar minimizar este aspecto, alguns professores começaram a utilizar técnicas que estão apresentando bons resultados. “Criar testes e jogos rápidos ao final de cada aula ajuda na permanência dos alunos, estimulando e mantendo o interesse no conteúdo” (BASTOS; BIAGIOTTI, 2014, p. 4). Desta forma, o desafio é descobrir formas de manter os participantes dos cursos motivados para que ocorra uma diminuição da evasão dos alunos, e consequentemente, um aumento da taxa de concluintes.

2.2 Gamificação

Ao longo dos últimos anos, a forma de educar sofreu diversas mudanças. Partiu-se de um modelo de aulas presenciais, com uma didática voltada ao professor para cursos híbridos e *online* que usam a tecnologia digital para fornecer aulas com uma pedagogia centrada no estudante e também com uma escala global. Porém, mesmo com o uso da tecnologia na educação para buscar uma melhora da qualidade do ensino, a motivação e o engajamento dos estudantes ainda é um problema que persiste na atualidade (KLOCK, et al., 2014).

Várias formas vêm sendo estudadas para aumentar a motivação e o empenho do estudante. Uma nova metodologia que está surgindo para tentar resolver este problema é a gamificação, ou pelo termo em inglês, *gamification*. Este conceito consiste na utilização de elementos dos jogos eletrônicos para motivar pessoas a realizar uma determinada ação (FARDO, 2013; KLOCK, et al., 2014).

2.2.1 Visão Geral sobre Gamificação

Segundo Fardo (2013), os jogos digitais, também conhecidos como *games*, são uma forma de entretenimento muito popular entre públicos das mais diversas idades. A gamificação, de uma maneira geral, é a utilização da capacidade de estímulo que estes jogos eletrônicos têm, para engajar e motivar pessoas a resolver problemas e auxiliar no aprendizado nas mais diversas áreas do conhecimento.

Seguindo essa linha, Klock et al. (2014, p. 2) define a gamificação como "o uso de mecanismos, estética e pensamento dos jogos para engajar pessoas, motivar ações, promover o conhecimento e resolver problemas". Já para Hamari, Koivisto e Sarsa (2014), o conceito de gamificação é definido como um processo que estimula e melhora a realização de serviços através do uso de recompensas que são fornecidas pelos *games*. Desta forma, este conceito pode ser definido como o uso dos elementos de jogos eletrônicos, em atividades que não são relacionadas aos *games*.

Fardo (2013) afirma que a gamificação faz uso de elementos normalmente encontrados nos jogos eletrônicos, como narrativa, sistema de realimentação, sistema de recompensas, competição, objetivos, níveis, diversão, e interação, com o intuito de conseguir fazer com que a motivação e o envolvimento para as atividades propostas seja a mesma que é alcançada quando existe uma interação com bons *games*.

Grande parte dos elementos dos jogos eletrônicos que são aplicados na gamificação estão diretamente relacionados aos desejos humanos, como pontuações e desafios. Desta forma, Klock (2014, p. 2) cita exemplos, relacionando com algumas necessidades do ser humano:

> [...] *pontos* são conectados com a necessidade de recompensa; *níveis* são úteis para demonstrar *status*; *desafios* permitem concluir realizações; *rankings* estimulam a competição; *presentes* deixam que as pessoas pratiquem a solidariedade (altruísmo).

Porém, a escolha dos elementos, e a forma que os mesmos serão aplicados depende muito da finalidade do projeto ou da atividade em questão.

> Podemos construir sistemas gamificados baseados apenas em pontos, medalhas e tabelas de líderes (PBL – Points, Badges and Leaderboards), que são apenas as mecânicas mais básicas de um game, com a finalidade única de promover mudanças no comportamento dos indivíduos, através de recompensas extrínsecas, semelhantes às ideias da economia comportamental, ou podemos construir uma experiência

> significativa que vá muito além do que as mecânicas básicas dos games oferecem e motivar intrinsecamente os indivíduos a desempenharem os seus papéis da melhor forma possível dentro do contexto em que se encontram (FARDO, 2013, p. 3).

Segundo Rapkiewicz et al. (2006), são os elementos e características dos *games* que os tornam atrativos, pois fornecem uma estrutura através de suas regras, além de possuírem metas que trazem motivação. Por serem interativos, permitem ação dos alunos e seus resultados favorecem o aprendizado, além de tornarem o processo agradável por serem divertidos.

2.2.2 Técnicas aplicadas na Gamificação

Segundo Roque et al. (2013), a gamificação proporciona aos utilizadores diversas características dos jogos, tais como: retorno imediato, repetição e pontuação. Para que isto seja possível, várias técnicas são utilizadas. A seguir, serão detalhadas as principais delas segundo o que foi descrito por Klock et al. (2014).

A técnica mais conhecida é o sistema de pontos, que tem um foco motivacional e funciona através da recompensa fornecida por pontuação. Vários são os tipos de pontos que podem ser utilizados, como por exemplo, os pontos de experiência, que também são conhecidos por XP, e normalmente são os mais importantes. Todas as ações geram pontos de experiência, e estes, podem guiar e classificar o usuário através de sua trajetória. Também existem os pontos resgatáveis, ou RP, que a medida em que são acumulados, podem ser trocados por itens de necessidade do usuário. Há também os pontos de habilidade, ou conhecidos no inglês pelo termo *skill*. São atribuídos a tarefas específicas, e são um conjunto extra de pontos em forma de bônus, que permite ao usuário ganhar experiência ou recompensa pela realização destas tarefas. Outros dois pontos podem ser citados: pontos de carma e reputação, que indicam respectivamente o nível de compartilhamento de informações e o quão confiável um usuário pode ser.

Outra técnica utilizada é a dos níveis, que de uma forma geral indica o progresso do usuário dentro da aplicação. Três diferentes tipos de níveis podem ser citados: níveis de jogo, níveis de dificuldade e níveis de jogador. O primeiro mostra qual a atual situação do jogo, e é focado em manter a percepção de que existe progresso dentro do sistema, além de desenvolver as habilidades do usuário a cada nível alcançado e motiva-lo a buscar novos desafios nos níveis mais difíceis. Os níveis de dificuldade ditam a complexidade dos desafios

propostos, e normalmente são separados em fáceis, médios e difíceis. O último tipo de nível, demonstram o progresso e a experiência do usuário, e são atribuídos conforme as tarefas forem sendo realizadas, os objetivos sendo alcançados e fidelização do usuário com o jogo.

Os *rankings* também são muito utilizados. Seu principal objetivo é a comparação entre usuários, pois mostra o progresso de todos os usuários dentro do ambiente, e com isso, instiga a competição entre eles. A classificação depende do desempenho de cada um, onde quanto melhor o usuário for, melhor classificado ele estará. Roque et al. (2013, p. 56) afirma que "os rankings criam rivalidade e aumentam o foco no objetivo. Jogadores competitivos estarão sempre em busca do primeiro lugar".

Seguindo o que foi descrito por Klock et al. (2014), a técnica dos desafios e missões são o que guiam os usuários sobre as atividades e objetivos que devem ser realizados. O uso dos desafios e das missões, faz com que as tarefas sejam interessantes, e motivem o usuário a realizar o que foi proposto. E para conseguir um empenho ainda maior, a técnica das medalhas pode ser colocada em prática. Esta, é uma versão de pontos mais robusta, que se trata de uma recompensa visual de alguma conquista.

2.3 Ensino de programação na atualidade

Em virtude deste trabalho ter o foco em ferramentas de auxílio no ensino de programação, esta seção será voltada no relato das formas atualmente usadas para ensinar algoritmos e programação, além das ferramentas que já existem nesta área.

As disciplinas de programação estão presentes em todos os cursos de computação e informática (JÚNIOR; RAPKIEWICZ, 2004). Segundo Rapkiewicz et al. (2006), matérias como estas abordam os princípios da lógica de programação, com o intuito de aperfeiçoar a capacidade de analisar e solucionar problemas.

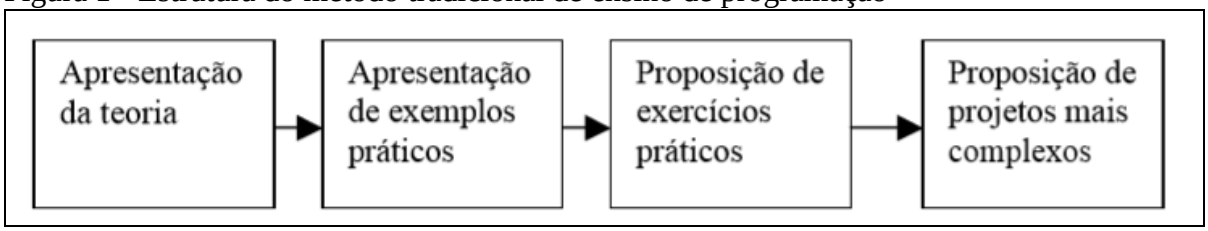
Porém, Júnior e Rapkiewicz (2004) comentam que é notável que grande parte dos estudantes de cadeiras de algoritmos e programação possuem dificuldades para entender e assimilar os conteúdos abordados nestas disciplinas, além de possuírem um alto índice de reprovação. Desta forma, comentam da necessidade de alterações nos métodos de ensino, para que possa existir uma melhora no processo de aprendizado.

2.3.1 Ensino tradicional de programação

Segundo Ferradin e Stephani (2005), o ensino tradicional de programação consiste em uma exposição do conteúdo da forma mais comum, ou seja, a teoria é apresentada, seguida

por exemplos básicos e a aplicação de exercícios simples, que vão aumentando sua complexidade com o passar das aulas. Nesta mesma linha, Borges (2000) explica que o modo tradicional utiliza uma postura didática que é comum em diversas áreas do conhecimento, apresentando conceitos de programação estruturada, como pode ser visto na Figura 1.

Figura 1 - Estrutura do método tradicional de ensino de programação



Fonte: BORGES, 2000, p. 1.

Normalmente, todo o ensino da área de programação inicia pela apresentação e utilização de algoritmos. “Um algoritmo consiste em um procedimento, composto por uma série de passos utilizados para resolver problemas computacionais específicos, que a partir do processamento com dados de entradas, irá gerar dados de saídas” (FERRADIN; STEPHANI, 2005, p. 3).

Porém, o ensino de algoritmos, segundo Souza (2009), não inicia com uma linguagem de programação propriamente dita, como Java ou C, mas sim, fazendo uso de pseudocódigos. O pseudocódigo, ou usualmente conhecido no Brasil como Portugol, é uma forma mais simples de escrever os códigos, porém com uma estrutura formal próxima as linguagens de programação. Assim, a resolução dos exercícios de algoritmos primeiramente são apresentadas com o uso de pseudocódigos, para depois, utilizar as linguagens de programação propriamente ditas.

Borges (2000) comenta que o modo tradicional causa desinteresse e desmotivação aos alunos, além de não deixar clara a importância destes assuntos para a sua formação. Rapkiewicz et al. (2006) afirma que as dificuldades em aprender a programar existem devido à falta de uma metodologia adequada de ensino, além da ausência de um material didático inovador.

2.3.2 Juízes online

Ferramentas que visam melhorar o atual processo de ensino de programação têm sido desenvolvidos nos últimos tempos. No âmbito de auxiliar na correção e avaliação de exercícios de programação, surgiram os juízes *online*. (CHAVES, et al., 2013).

Grande parte dos programas de natureza algorítmica necessita receber como entrada apenas um padrão formatado de dados e, a partir desta entrada, realizar o processamento dos dados. Após processados, os resultados são obtidos e apresentados também de forma padronizada. Sendo assim, é possível que programas com esta característica sejam avaliados automaticamente através de uma ferramenta que gere os dados de entrada e de outra que capture os dados de saída e faça um comparativo para chegar a uma conclusão (CHAVES, et al., 2013).

Chaves et al. (2013), explica que os sistemas que realizam automaticamente o processo de avaliação destes programas são conhecidos por juízes *online*, ou pelo termo em inglês *online judges*. Em sistemas como este, um programa é submetido, e seu código-fonte é compilado, executado e testado para verificar se está escrito corretamente. Para os testes, o sistema utiliza dados formatados como a entrada, processa os dados e compara os resultados obtidos com os resultados esperados, fornecendo uma resposta adequada com base nestas comparações, como por exemplo se o programa está certo ou errado, e até se tem problemas de execução ou compilação.

Várias destas ferramentas são facilmente encontradas na Internet. Chaves et al. (2013) cita dois *sites* populares: o SPOJ Brasil e o URI *Online Judge*. Estes sistemas disponibilizam vários problemas para serem resolvidos e submetidos, bastando o usuário selecionar a linguagem de programação a ser utilizada no desenvolvimento da atividade e enviar a solução para ser avaliada. Fóruns de discussão, estatísticas sobre os problemas e uma lista de classificação dos usuários também são disponibilizadas. Santos e Ribeiro (2011) citam também o UVA Online Judge, que é uma outra ferramenta bastante conhecida nesta área.

Porém, em grande parte das vezes, estes juízes *online* têm seu desenvolvimento voltado ao uso em competições de programação e maratonas de desenvolvimento. E neste caso, estas ferramentas não disponibilizam muitas funcionalidades didáticas. Um exemplo que pode ser citado é sobre o retorno dado ao usuário no momento da submissão de uma atividade ou exercício. As informações fornecidas, conforme mencionado anteriormente, somente contemplam dados como correto ou incorreto e erros de compilação. Porém, resultados como este são pouco instrutivos e não auxiliam a localizar erros, sendo responsabilidade da pessoa localizar as falhas e corrigi-las (SANTOS; RIBEIRO, 2011).

Apesar destas ferramentas terem grande utilidade para testes e principalmente, correção de programas e exercícios, seu uso para fins didáticos é limitado. Outras formas devem ser pesquisadas e estudadas para melhorar o ensino de programação (SANTOS; RIBEIRO, 2011).

2.3.3 Ferramentas de autoestudo de programação

Também na área de auxílio ao ensino de programação, surgiram ambientes *online* que além de avaliarem os programas escritos por aluno, oferecem uma infraestrutura com diversas atividades e exercícios que podem ser resolvidos dentro própria plataforma, e fornecem realimentação para os estudantes, auxiliando tanto no aprendizado do aluno, quanto otimizando o tempo do professor. A avaliação dos programas submetidos por alunos é mais avançada, podendo levar em conta diversos fatores, como indentação do código, legibilidade, assertividade, entre outros (CAIZA; ALAMO, 2013).

Segundo Caiza e Alamo (2013), algumas destas ferramentas são mais antigas e maduras, fornecendo uma estrutura mais robusta e completa. É o caso de ferramentas como CourseMarker, Virtual Programming Lab e WebCat. Outros sistemas mais recentes também foram estudados, ainda que não sejam muito utilizados, contudo apresentando novas funcionalidades e novas linhas de pesquisa. Detalhes sobre algumas das ferramentas mais completas serão descritos a seguir, segundo Caiza e Alamo (2013) e Ihantola et al. (2010).

O CourseMarker é uma plataforma para avaliação automática de exercícios, que possui como linguagens de programação suportadas o Java e C++. Possui uma estrutura em cursos, que faz com que os exercícios sejam divididos em áreas específicas, permitindo mostrar ao aluno o progresso das atividades. As avaliações dos programas levam em conta tipografia (indentação, comentários, etc.), funcionalidade por testes de caso, estrutura do programa e verificação dos objetos utilizados no código, além de possuir detecção de plágio.

A plataforma Virtual Programming Lab (VPL) é uma ferramenta que tem como foco a atribuição de atividades para os alunos e a avaliação dos exercícios submetidos. Suporta diversas linguagens, como C, C++, Java, PHP e Python. A arquitetura foi desenvolvida como um *plugin* para o Moodle, permitindo que as submissões e avaliações fossem realizadas do próprio ambiente *online*. Suas avaliações consideram testes de caso, que são especificados pelo professor por uma sintaxe da própria ferramenta. É uma ferramenta aberta, podendo ser usada e alterada por qualquer pessoa que tem interesse.

WebCat também é uma ferramenta *opensource* que faz avaliação de exercícios. Neste ambiente, os próprios alunos planejam os testes unitários, ou seja, os estudantes desenvolvem os testes que o programa deverá realizar, e a nota dependerá dos resultados obtidos nestes testes. O WebCat também armazena o progresso do aluno, e exibe em forma de gráficos a evolução dos alunos, mostrando em cada exercício os erros e acertos. Suporta linguagens como Java, C++ e Pascal.

Algumas das ferramentas mais recentes descritas pelos autores são o JavaBrat, AutoLEP e Petcha. Porém, uma nova ferramenta que não está nos artigos mencionados será comentada por buscar integrar diversas funcionalidades que os principais ambientes desta área possuem. O CodeWorkout está em fase de desenvolvimento, e está sendo criado pela mesma equipe de desenvolvedores do WebCat (CODEWORKOUT, 2015).

O CodeWorkout, diferentemente dos ambientes citados anteriormente, não tem o foco em programas maiores, mas sim, voltado a exercícios menores, como o desenvolvimento de funções e métodos. O objetivo de utilizar atividades em uma escala menor é proporcionar um grande número de pequenos exercícios com temas diversificados, fazendo com que os alunos pratiquem um pouco de cada habilidade nas diferentes atividades (CODEWORKOUT, 2015).

Além desta nova metodologia, segundo o site CODEWORKOUT (2015), a ferramenta proporciona também as avaliações automáticas fornecidas pelas ferramentas anteriores, além de questões de múltipla escolha, e fornecimento de dicas para alunos que não estão conseguindo dar andamento nas atividades.

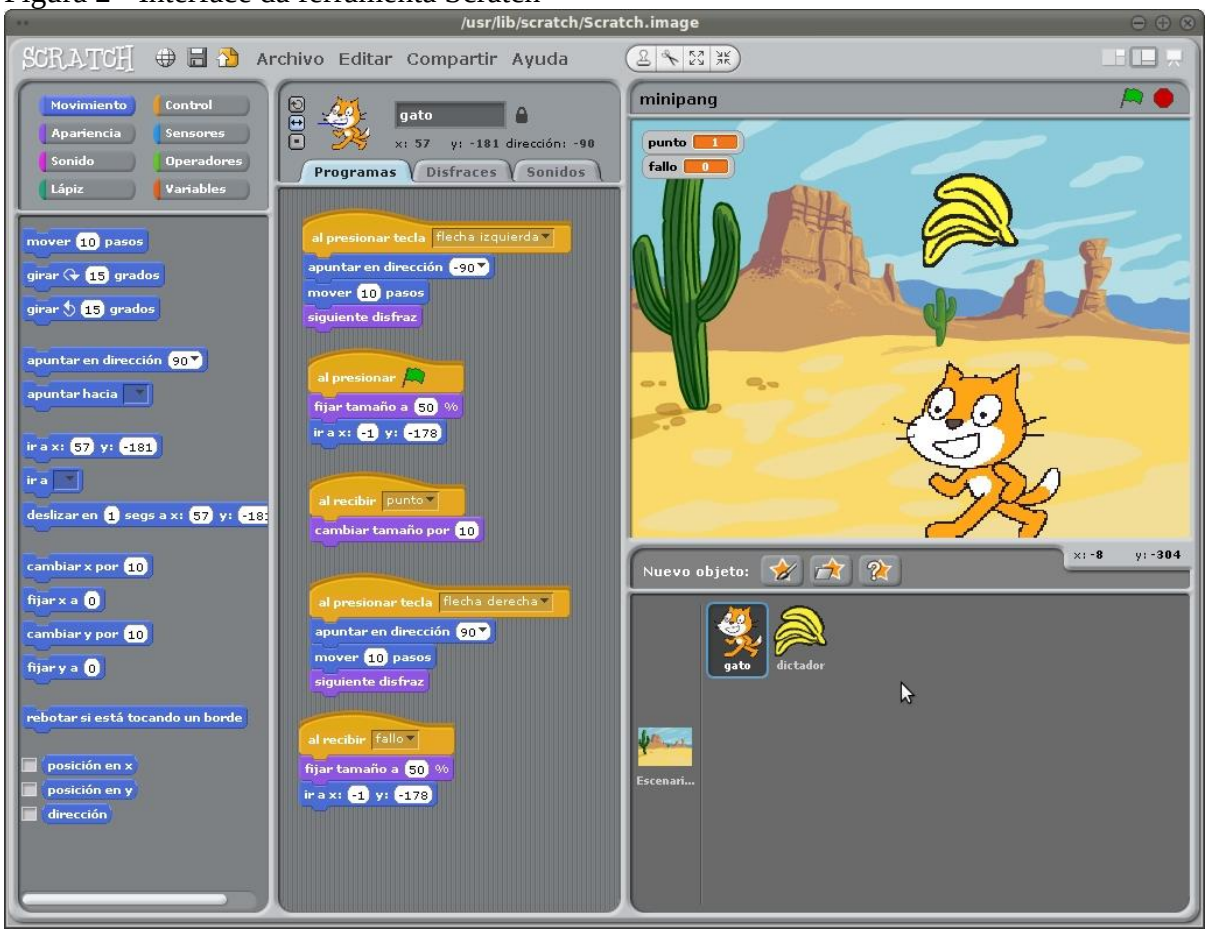
2.3.4 Ferramentas de auxílio à programação através do desenvolvimento de jogos

Com o intuito mais motivacional, surgiram ambientes de aprendizado que usam o desenvolvimento de jogos para atrair atenção dos alunos, como o Scratch (SCRATCH, 2015), Alice (ALICE, 2015) e Robocode (ROBOCODE, 2015).

O Scratch é um ambiente que foi desenvolvido pelo Lifelong Kindergarten Group, com o intuito de ensinar programação de uma maneira fácil e rápida, para todos que tem pouco, ou nenhum conhecimento nesta área. Para isso, utiliza uma linguagem de programação visual, que permite a manipulação de objetos e mídia, para criação de jogos e animações (AURELIANO; TEDESCO, 2012). Segundo o site SCRATCH, o ambiente foi projetado inicialmente para pessoas entre 8 e 16 anos, mas hoje é utilizado por pessoas de todas as idades.

A programação neste ambiente é baseada em blocos, que através do encaixe e empilhamento destes, formam as estruturas básicas de um programa. Cada bloco possui comandos já estabelecidos, e através da montagem adequada destes, a lógica começa a ser desenvolvida, e com isso, o programa passa a funcionar. Os comandos precisam apenas serem arrastados e soltados no espaço dos comandos, onde os mesmos são encaixados uns nos outros (AURELIANO; TEDESCO, 2012; MÉLO et al., 2011). Uma imagem da interface do Scratch pode ser vista na figura 2.

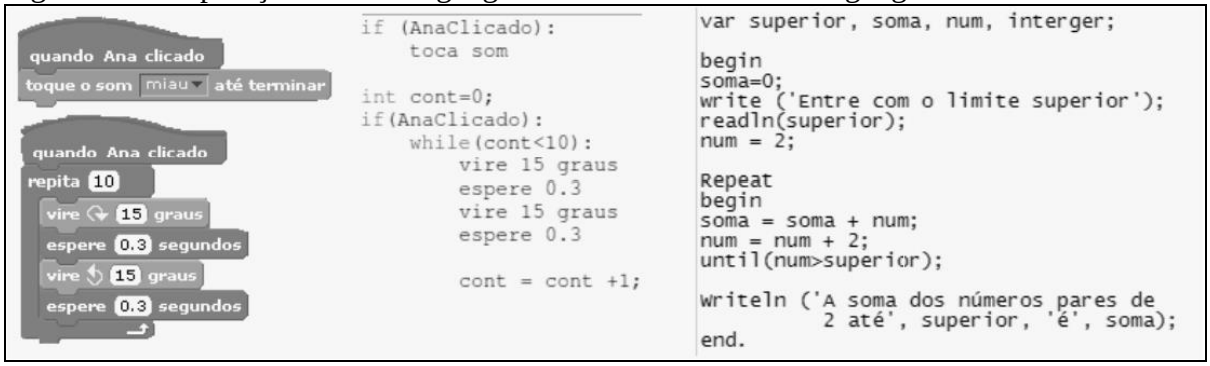
Figura 2 - Interface da ferramenta Scratch



Fonte: Ferramenta Scratch.

Segundo Scaico et al. (2012), esta ferramenta melhora o aprendizado pois utiliza um conceito inovador, conhecido como desenvolvimento orientado ao *design*. Devido à sua estrutura em blocos, a ferramenta se torna mais didática, auxiliando no aprendizado. Uma comparação entre a estrutura do Scratch e a sintaxe de duas linguagens de programação pode ser vista na Figura 3.

Figura 3 - Comparação entre a linguagem Scratch com demais linguagens



Fonte: SCAICO et al., 2012, p. 3.

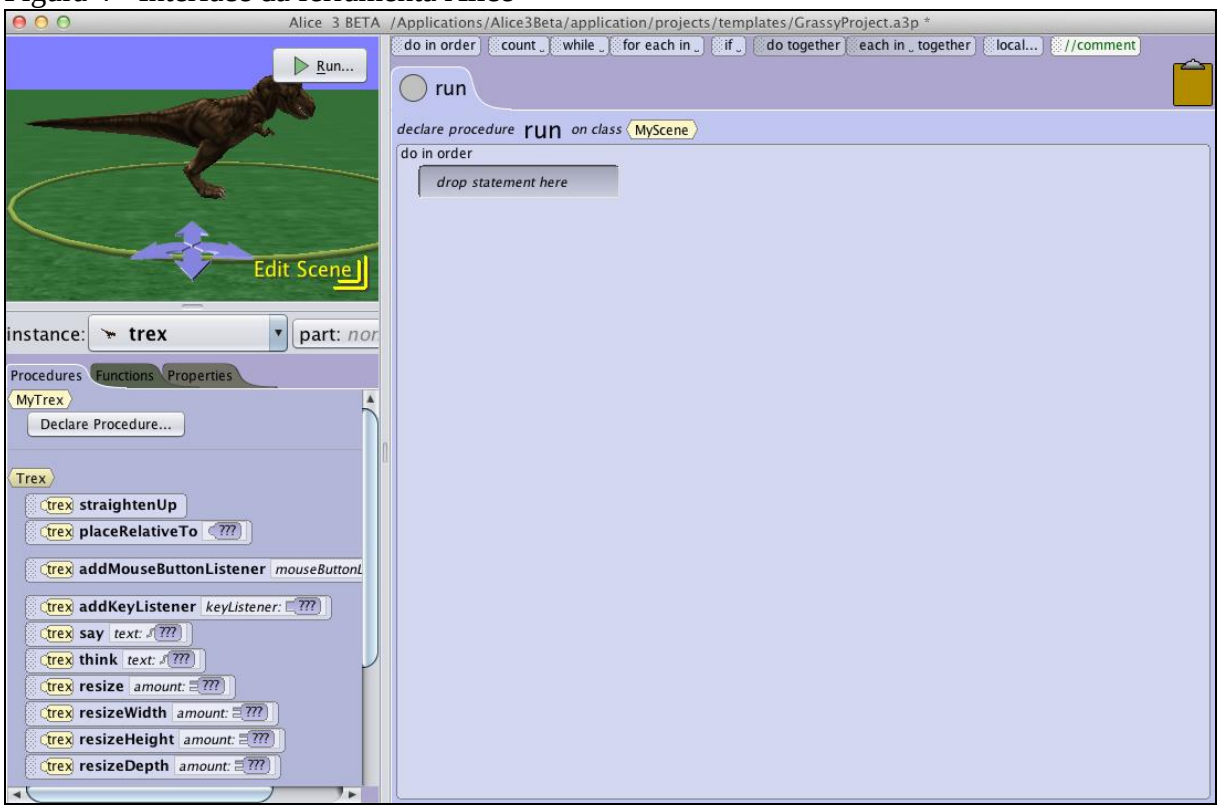
A ferramenta faz com que o aluno foque sua atenção no desenvolvimento da lógica, não prendendo sua atenção em assuntos que não são de grande importância no início do aprendizado. Assim, Scaico et al. (2012, p. 3) afirma que “o caráter mais didático do Scratch projeta no aluno a possibilidade dele se concentrar no exercício do pensamento algorítmico e na criatividade para a construção das soluções”.

Outra ferramenta que segue esta linha é Alice, que segundo o site ALICE (2015) é um ambiente de desenvolvimento 3D que facilita a criação de animações ou de jogos. Esta ferramenta foi desenvolvida para auxiliar estudantes a aprender conceitos fundamentais de programação, além de introduzir conceitos de programação orientada a objetos.

Nesta ferramenta, assim como no Scratch, o conceito de arrastar e soltar comandos também é utilizado, entretanto, os objetos estão situados em um mundo virtual tridimensional.

Dann et al. (2012) comentam que Alice é uma boa ferramenta para programadores iniciantes, já que o ambiente permite aos usuários a possibilidade de ver imediatamente como o programa desenvolvido está sendo executado, permitindo um fácil entendimento da relação entre os comandos e as declarações com o comportamento dos objetos nas suas animações. A Figura 4 mostra a interface da ferramenta Alice.

Figura 4 - Interface da ferramenta Alice



Fonte: Ferramenta Alice.

Além das citadas anteriormente, o Robocode é outra ferramenta que usa o desenvolvimento de jogos para incentivar e ensinar pessoas a programação, sendo o Java a linguagem principal. Segundo o *site* ROBOCODE (2015), o Robocode é um jogo de programação, onde o principal objetivo é programar um tanque para participar de uma competição.

Contudo, além de ser um jogo, a plataforma também é utilizada para a prática de programação. Escolas e universidades estão utilizando a ferramenta como parte do ensino de programação, devido ao conceito ser de fácil entendimento, além de uma forma divertida de aprender (ROBOCODE, 2015).

Figura 5 - Batalha entre tanques na ferramenta Robocode

Fonte: Ferramenta Robocode.

2.4 Gamificação no ensino de programação

Como pôde ser visto anteriormente, ensinar a programar é considerada uma tarefa difícil, e a forma que vem sendo realizada muitas vezes é ineficiente. Em virtude disso, várias formas de ensinar estão sendo pesquisadas para tentar melhorar e facilitar o entendimento dos

algoritmos e da programação. Uma destas formas é a utilização da gamificação, já que oferece uma variedade de maneiras para aumentar o interesse dos alunos no aprendizado (SWACHA; BASZURO, 2013).

Segundo Rapkiewicz et al. (2006), vários problemas podem ser citados no ensino de algoritmos e programação, onde a dificuldade em desenvolver raciocínio lógico e a falta de motivação em virtude da dificuldade podem ser destacados. Neste contexto, os autores citam que durante os *games*, os jogadores pensam da mesma forma que os programadores raciocinam enquanto escrevem um código.

> Podemos dizer que o algoritmo está presente implicitamente no jogo e no modo como os alunos pensam no jogo, só que o aluno ainda não consegue enxerga-lo explicitamente. Através dos jogos, os alunos se sentem mais motivados no desenvolvimento das tarefas e, consequentemente, desenvolvem o raciocínio, sobretudo quando os jogos agregam um conjunto de elementos multimídia que prendem mais a atenção do que questões em papel ou no quadro (RAPKIEWICZ et al., 2006, p. 4).

Desta forma, Rapkiewicz et al. (2006) reforça que dentro do ensino de programação, os jogos podem fornecer recursos além do entretenimento e da diversão. Os autores comentam que pelo fato dos *games* forçarem o aluno a decidir, escolher e priorizar, ocorre uma melhora no desenvolvimento do raciocínio lógico, que é um elemento fundamental para quem está aprendendo a programar.

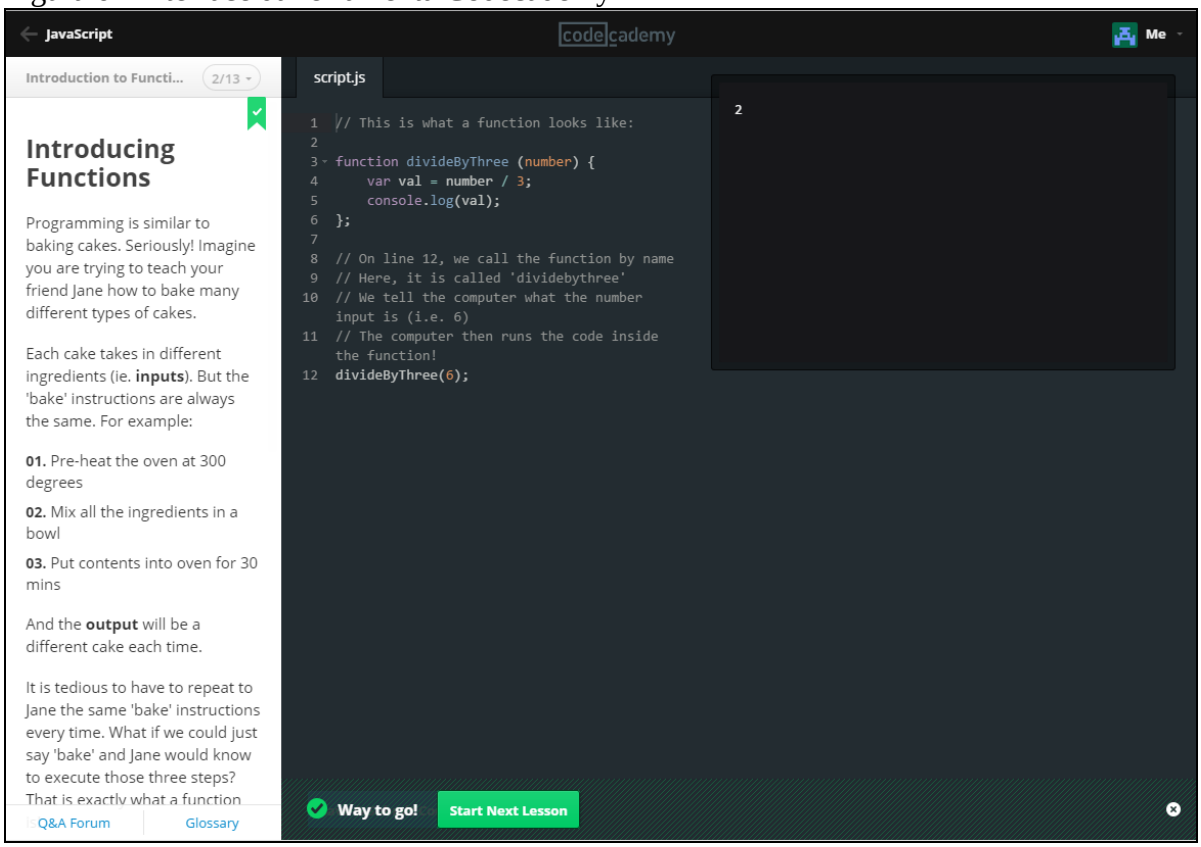
As técnicas de gamificação podem ser aplicadas até mesmo no processo tradicional de ensino, porém, Baszuro e Swacha (2013) afirmam que as formas de aprender e ensinar devem ser focadas em ambientes *online*, em virtude do crescimento da demanda na educação pela Internet. Assim, com o uso da tecnologia e de técnicas como a gamificação, diferentes metodologias podem ser usadas mudar a maneira que os algoritmos e a programação são ensinados atualmente.

2.4.1 Ambientes gamificados de ensino de programação

Dentro do ensino de programação, algumas ferramentas com conceitos de gamificação já foram desenvolvidas. Dentre elas, é possível citar o Codecademy (CODECADEMY, 2015), o CoderByte (CODERBYTE, 2015), Khan Academy (KHANACADEMY, 2015), e Learneroo (LEARNEROO, 2015).

Codecademy é uma ferramenta especializada no ensino de programação, e projetada com conceitos de gamificação. Os cursos são organizados e separados em seções, e consistem em sequencias de exercícios. Estes exercícios são compostos em um texto introdutório, instruções que explicam ao estudante o que fazer, e um local interativo onde a atividade deve ser realizada (SWACHA; BASZURO, 2013). Na Figura 6 é possível observar a interface do ambiente Codecademy.

Figura 6 - Interface da ferramenta Codecademy



Fonte: Site CODECADEMY, 2015.

Esta ferramenta utiliza diversas técnicas de gamificação. Swacha e Baszuro (2013) citam que pontos são fornecidos aos estudantes para cada exercício concluído. Além disso, os estudantes são recompensados por medalhas ou *badges* quando completam lições ou conseguem realizar tarefas mais difíceis.

Segundo o site do Codecademy, diversas linguagens de programação estão disponíveis para estudar. Alguns exemplos que podem ser encontrados são o Python, Ruby, e Javascript (CODECADEMY, 2015).

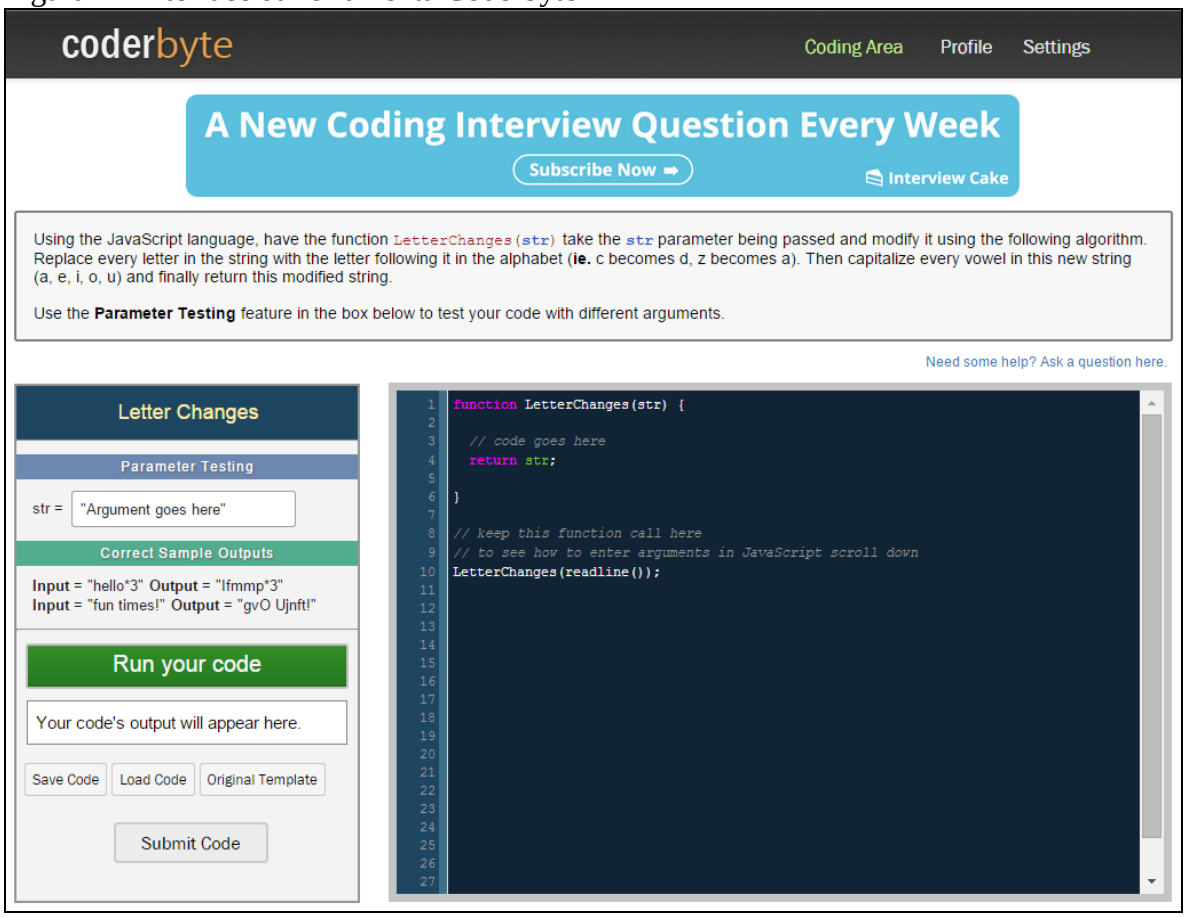
Outra ferramenta que também utiliza conceitos de gamificação é o Coderbyte. Este ambiente visa a pratica de programação, com uma biblioteca de desafios em forma de exercícios, que são classificados em níveis de dificuldade. Pontos também são fornecidos aos

usuários. Entretanto, nesta ferramenta, a pontuação depende dos testes de uso e do tempo que o usuário demorou para solucionar o problema (CODERBYTE, 2015).

Além dos pontos, o conceito de recompensar através de medalhas também é utilizado nesta ferramenta. O Coderbyte também utiliza a técnica de níveis, visto que todos os exercícios são classificados por dificuldades diferentes (CODERBYTE, 2015).

Uma imagem da ferramenta pode ser vista na Figura 7.

Figura 7 - Interface da ferramenta Coderbyte

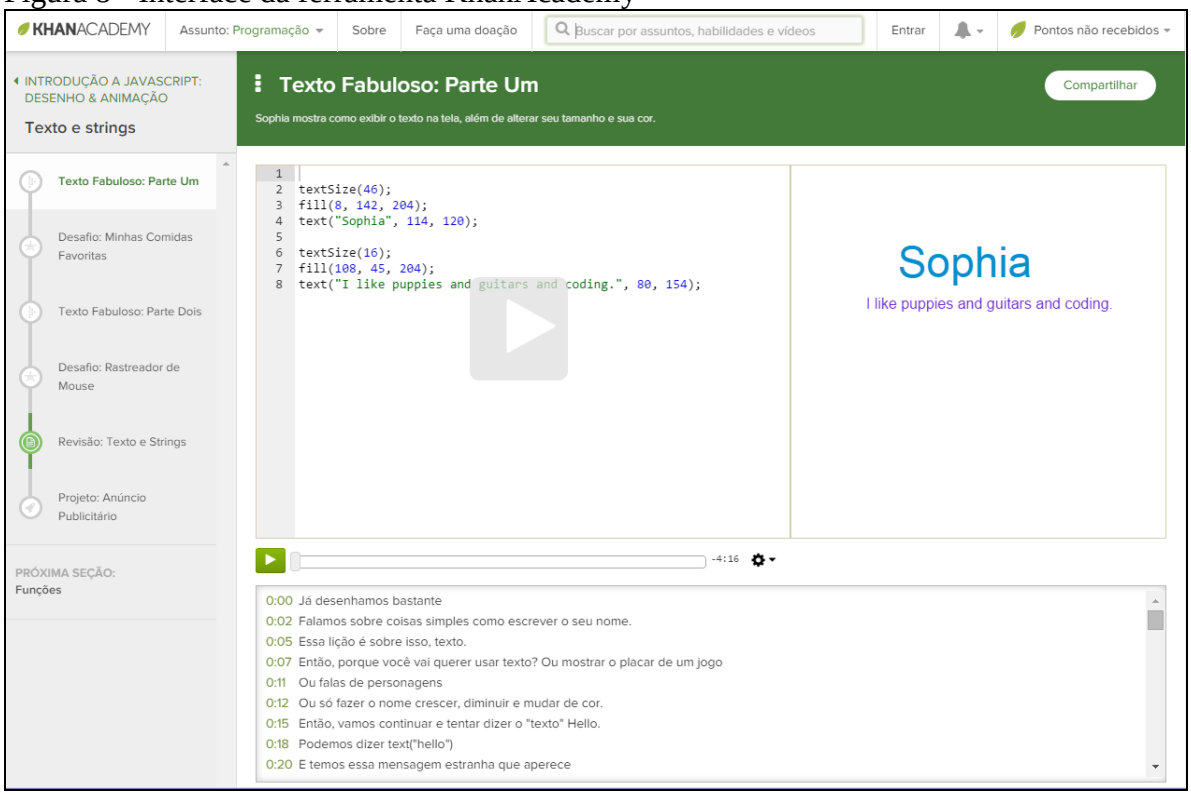


Fonte: Site CODERBYTE, 2015.

O Khan Academy é um ambiente gamificado de ensino que contempla diversas áreas do conhecimento, incluindo Matemática, Ciências, Computação entre outros. Dentro da computação, o ensino de programação é realizado (KHANACADEMY, 2015).

Os conceitos principais da gamificação também estão incluídos nesta ferramenta, como uso de medalhas, visualização de progresso e pontos. As aulas são divididas em diversos exercícios, que são realizados seguindo um caminho, semelhante ao que é feito no Codecademy. Vídeos também são disponibilizados, além de dicas que podem facilitar o entendimento do assunto.

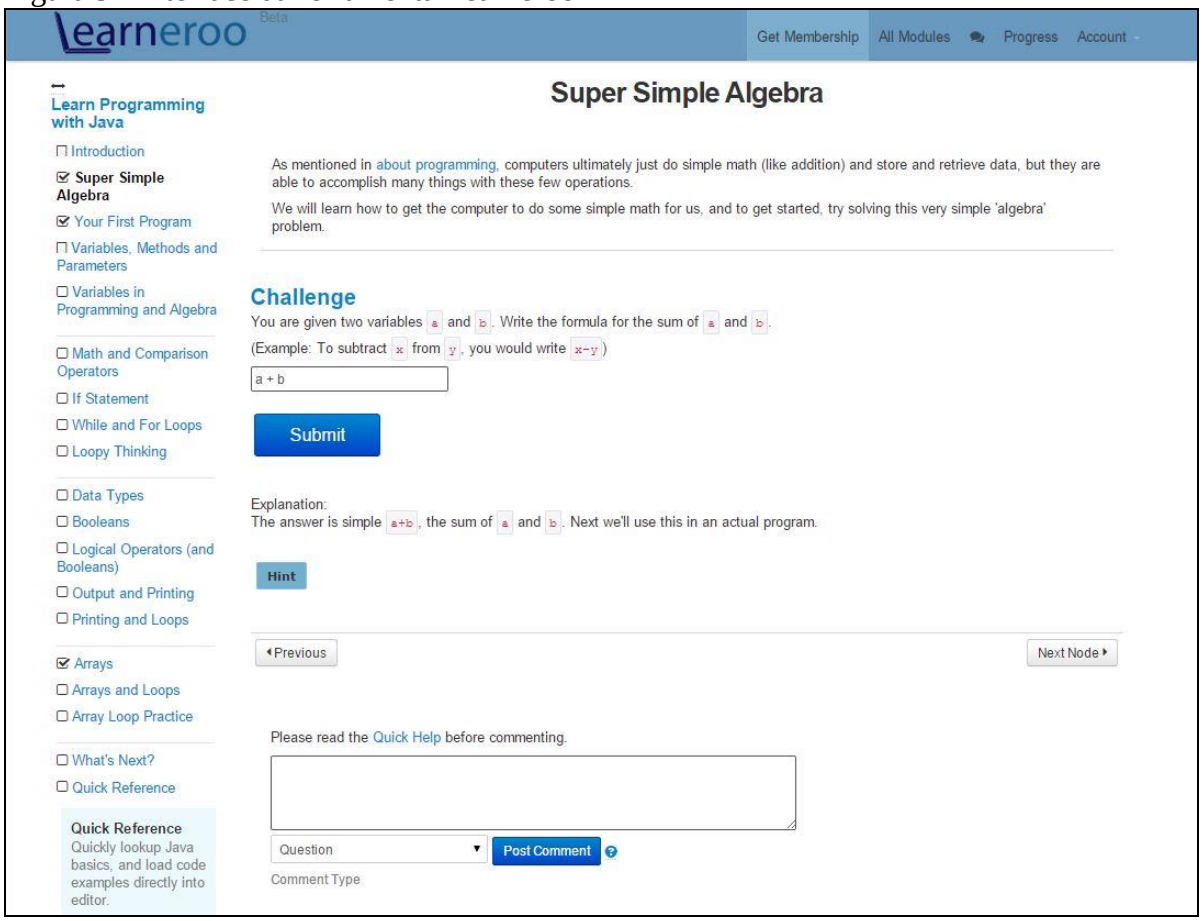
Figura 8 - Interface da ferramenta KhanAcademy



Fonte: Site KHANACADEMY, 2015.

Outra ferramenta gamificada é o Learneroo. Esta ferramenta possui uma gama de exercícios tanto de múltipla escolha, quanto de programação de pequenos módulos. As atividades seguem uma ordem, e são estruturadas em instruções e explicações iniciais, um espaço para responder as perguntas ou para realizar o desenvolvimento do desafio, e um local onde dicas podem ser solicitadas pelos alunos que não conseguem completar o exercício. O ensino é realizado através de tutoriais nos próprios exercícios, e utiliza técnicas gamificadas de *leaderboard* além de mostrar o progresso do aluno dentro de cada módulo (LEARNEROO, 2015). A interface da ferramenta pode ser vista na Figura 9.

Figura 9 - Interface da ferramenta Learneroo



Fonte: Site LEARNEROO, 2015.

2.5 Geradores de exercícios de programação

Conforme pôde ser visto anteriormente, diversas ferramentas que auxiliam no ensino e no estudo de programação já foram ou estão sendo desenvolvidas. O uso da tecnologia está favorecendo a criação de ambientes que auxiliam no aprendizado de programação, além de aumentar a disseminação do conhecimento. Novas metodologias de ensino, como a gamificação, também estão sendo utilizadas, para incentivar o aprendizado e a busca pela informação.

Porém, foi verificado que ainda não foi desenvolvida uma ferramenta capaz de gerar exercícios de programação automaticamente. Os ambientes mencionados acima que disponibilizam atividades para os alunos, possuem uma grande lista estática de exercícios, fazendo com que todos os problemas fornecidos sejam iguais para todos os usuários.

Desta forma, uma ferramenta que possa gerar e diversificar exercícios seria de grande valia para estes ambientes, já que as atividades disponibilizadas para os alunos seriam diferentes umas das outras, fazendo com que cada estudante tenha que resolver seu próprio

problema. Além disso, uma ferramenta que possibilite a geração de exercícios poderia ser útil para professores que desejam aplicar testes ou criar listas de exercícios para um grande número de estudantes.

Assim, a proposta deste trabalho será desenvolver uma ferramenta que possa gerar exercícios de programação de forma automática, visando auxílio para ensinar e praticar atividades de desenvolvimento.

3 DESENVOLVIMENTO

Este capítulo tem como objetivo descrever como foi realizada a implementação do sistema gerador de exercícios[1]. Nos capítulos a seguir, serão detalhadas as metodologias empregadas bem como as ferramentas e bibliotecas necessárias para conclusão do desenvolvimento.

3.1 Organização do sistema

O sistema foi desenvolvido através de três componentes principais: o gerador, o tradutor e uma lista de *templates*. Através destes, o sistema possibilita a geração de uma lista de exercícios, onde cada um pode conter, dentre outras saídas, o título, um enunciado e sua solução em até quatro linguagens de programação, sendo elas Java, C, C++ e Python.

O software funciona com base nos *templates*, que são arquivos estruturados e que contém informações sobre o próprio *template*, como o nível de dificuldade do exercício e seu tópico principal, juntamente com um enunciado genérico e sua solução em formato de pseudocódigo. Cada *template* equivale a um exercício, e em seu texto são contidas lacunas para serem preenchidas aleatoriamente, possibilitando a criação de exercícios diversificados. O exercício de saída é criado com base no texto do *template*, ou seja, o conteúdo gerado na saída é uma modificação aleatória do texto original do arquivo.

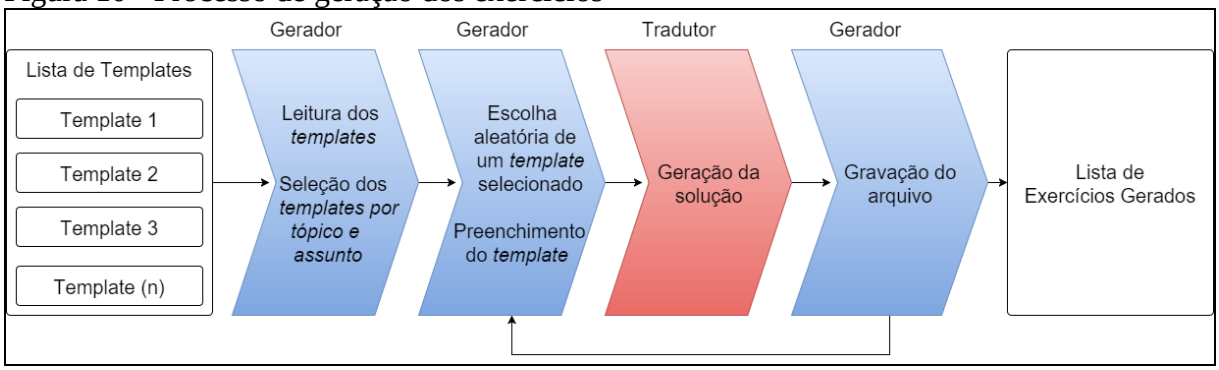
Inicialmente, o sistema recebe parâmetros do usuário, como a quantidade e o assunto principal dos exercícios, além da linguagem de saída para a solução. Com estes dados, o módulo gerador faz a leitura de uma lista de *templates* e os seleciona de acordo com os critérios do usuário. Após esta etapa, seleciona aleatoriamente um dos *templates*, preenche com dados também aleatórios, e o envia para o módulo tradutor, que por sua vez, gera sua solução na linguagem solicitada. Com a solução gerada, retorna ao gerador para realizar a

[1] O sistema desenvolvido foi documentado em inglês e disponibilizado como código aberto no Github, podendo ser acessado pelo *link* (https://github.com/ptramontina/exercise_generator).

gravação do arquivo. Este processo é realizado até que o número de exercícios informado seja alcançado.

A Figura 10 demonstra de uma forma geral o processo utilizado pelo sistema para geração dos exercícios.

Figura 10 - Processo de geração dos exercícios



Fonte: Elaborado pelo autor.

Para gerar a solução, o módulo tradutor faz uso do pseudocódigo contido dentro do *template*. Este pseudocódigo é a solução do exercício, e tem sua estrutura semelhante ao Portugol, porém, com termos criados na língua inglesa. A proposta de usar um pseudocódigo para gerar a solução do exercício foi utilizada com o intuito de auxiliar no ensino de programação. Desta forma, o pseudocódigo pode ser disponibilizado ao aluno nos casos em que estiver com dificuldades para resolver o problema, tornando a solução mais didática. Além disso, com o processo de geração da solução tendo como base o pseudocódigo, tornou mais simples disponibilizar a solução em diferentes linguagens de programação, já que a saída é gerada com base nos termos lidos do pseudocódigo.

Como o sistema tem a proposta de auxiliar no ensino de programação, além do pseudocódigo, cada exercício pode ter em sua saída, dicas e resultados esperados, com a finalidade de auxiliar o aluno no aprendizado e no entendimento do exercício.

Nos tópicos a seguir, serão detalhados os desenvolvimentos dos módulos, bem como a integração entre eles.

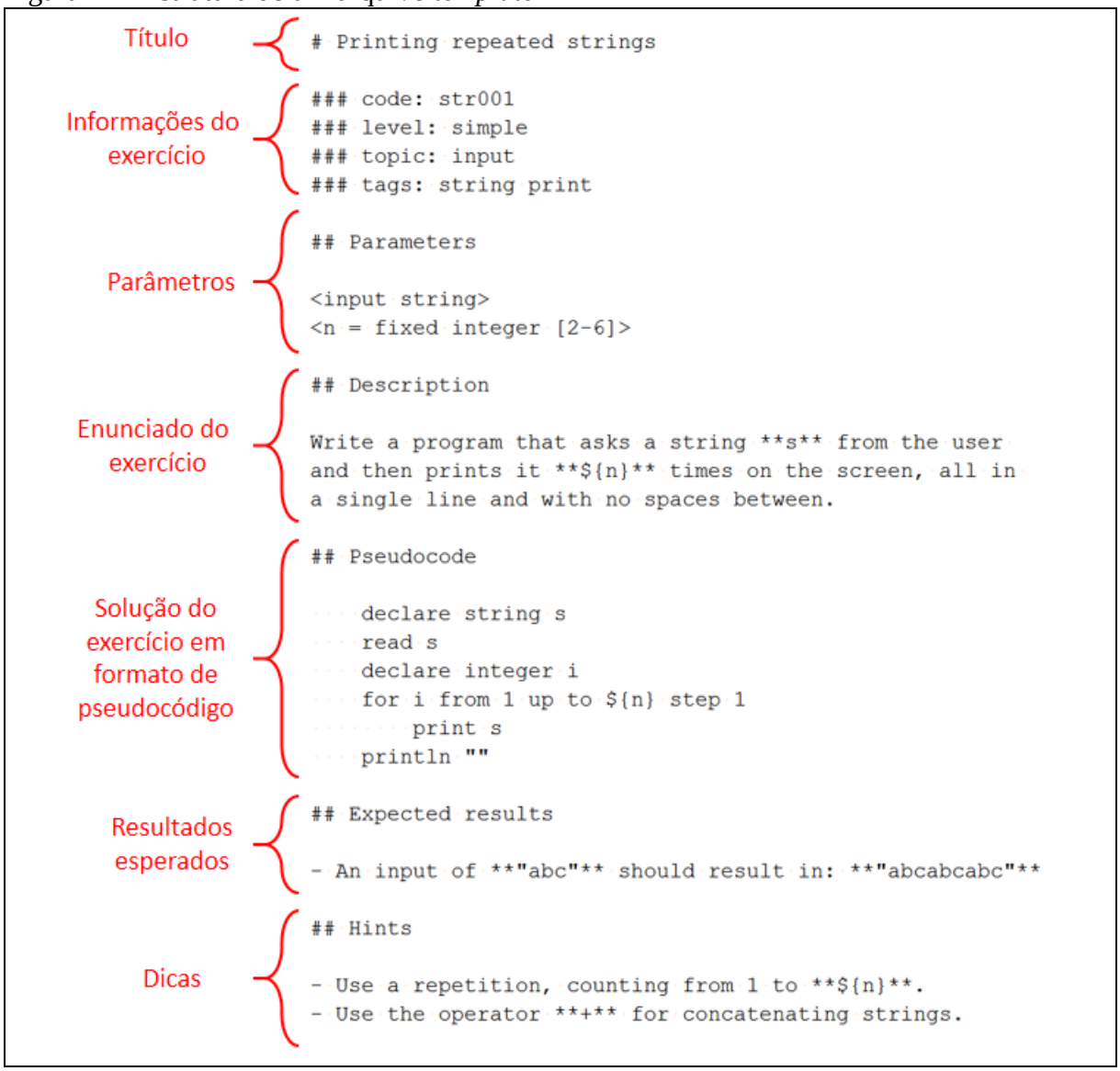
3.2 Template

O *template* é basicamente um arquivo de texto que possui todas as informações necessárias para gerar um exercício completo. Este arquivo deve ser criado de forma manual, e ser previamente adicionado no local onde estão contidos todos os *templates*.

Sua estrutura segue o formato de uma linguagem de marcação, mais especificamente usando a sintaxe Markdown. A proposta de utilizar uma linguagem como esta ocorreu em virtude da saída ser gerada pronta para uso. Desta forma, o arquivo já é criado com uma formatação básica, que provém da própria linguagem de marcação, bastando apenas disponibilizar o exercício criado em um ambiente que formata o texto com base nesta sintaxe.

A estrutura do arquivo deve seguir algumas regras para que funcione adequadamente. Para um melhor entendimento, a Figura 11 demonstra a estrutura de um *template*.

Figura 11 - Estrutura de um arquivo *template*



Fonte: Elaborado pelo autor.

Na sintaxe da linguagem Markdown, o caractere “#” é utilizado no início da linha para formatar o texto como título. Assim, dependendo da quantidade destes caracteres inseridos no

início da linha, será definido o tamanho da fonte de saída. Por exemplo, se o caractere "#" for utilizado apenas uma vez, indica um título de nível um, ou seja, a fonte de maior tamanho. Caso seja utilizado três vezes, indica um título de terceiro nível (DARING FIREBALL, 2015).

O *template* deve seguir essa estrutura para que o módulo gerador possa identificar cada parte do texto e separa-las adequadamente. Cada seção é separada pelos níveis de título da sintaxe Markdown, juntamente com o nome escrito em inglês, com exceção do título, que não está descrito no início da seção. Cada parte será detalhada nos tópicos a seguir.

3.2.1 Título

O título descreve o nome do exercício, e será exibido com a maior fonte, na parte superior do texto. Esta seção é definida pelo nível um da sintaxe Markdown, e portanto, deve iniciar com o caractere "#", seguido de um espaço.

3.2.2 Informações do exercício

As informações do exercício são compostas pelo código do *template*, o nível de dificuldade da atividade, o tópico principal e pelos rótulos do exercício, definidas respectivamente pelas palavras em inglês *code*, *level*, *topic* e *tags*. O código é utilizado para diferenciação de cada *template*. O nível indica qual a complexidade do exercício em questão. O tópico indica o assunto principal do exercício, ou seja, até qual conteúdo o *template* aborda. Já as *tags* indicam os outros assuntos que estão contidos nesta atividade.

Para serem interpretados corretamente, cada uma das palavras em inglês descritas anteriormente devem ser precedidas pelos caracteres "###" seguidas de um espaço.

3.2.3 Parâmetros

Os parâmetros do *template* são inseridos abaixo do termo *parameters* precedidos dos caracteres "##" e de um espaço. Os parâmetros podem ser de dois tipos: de entrada e de preenchimento. O primeiro indica que tipo de dados o exercício recebe como entrada. Atualmente, o sistema apenas armazena este tipo de parâmetro, porém, não utiliza para realizar a geração dos exercícios. Já os parâmetros de preenchimento são utilizados pelo módulo gerador para inserir dados aleatórios nas lacunas dos *templates*.

Cada parâmetro de preenchimento é inserido em uma linha e delimitado pelos caracteres "<" e ">". Estes, devem conter o nome da variável, seguido pelo sinal de

igualdade, o tipo de variável e o intervalo do valor de sorteio entre colchetes. Por exemplo, a sintaxe "**<n = fixed integer [2 – 6]>**" indica que onde houver a variável "**n**" no texto, deverá ser substituída por um inteiro entre dois e seis, selecionado aleatoriamente. A Tabela 1 demonstra todas as possibilidades que podem ser utilizadas como parâmetros de preenchimento.

Tabela 1 - Detalhamento dos parâmetros do sistema

Parâmetro	Descrição do parâmetro
<n = fixed integer [2 – 6]>	Variável "n" recebe um inteiro entre 2 e 6.
<c = fixed char [a – z]>	Variável "c" recebe um caractere entre 'a' e 'z'.
<d = fixed double [0.0 – 2.0]>	Variável "d" recebe um número real entre 0.0 e 2.0.
<s = fixed string ["abc", "def", "ghi"]>	Variável "s" recebe uma das strings dentro dos colchetes.
<b = fixed boolean >	Variável "b" recebe verdadeiro ou falso.

Fonte: Elaborado pelo autor.

3.2.4 Descrição

A descrição se refere ao enunciado genérico da atividade, ou seja, esta seção contém a descrição do exercício que será criado. Nesta, o criador do *template* deve descrever o que será solicitado no exercício e informar os locais onde deseja que os valores sejam substituídos pelos dados aleatórios. Para isto, devem ser inseridos nos lugares desejados uma sintaxe que inicia com "**${**", juntamente com o nome de uma das variáveis contidas nos parâmetros de preenchimento, e finalizando com "**}**".

3.2.5 Solução em formato de pseudocódigo

Nesta seção, está contida a resposta para o enunciado descrito anteriormente. Porém, esta solução deve seguir o formato do pseudocódigo desenvolvido para este sistema. Este pseudocódigo é semelhante às construções de uma linguagem de programação convencional, porém, em uma escala reduzida, e com um número menor de comandos. As expressões não levam o ponto e vírgula no final e os blocos são delimitados pela indentação, através da adição de quatro espaços para cada nível. A Figura 12 faz um comparativo da sintaxe do pseudocódigo proposto com a linguagem Java.

Figura 12 - Comparativo do pseudocódigo criado e a sintaxe Java

PSEUDOCÓDIGO	JAVA
TIPOS	
boolean	boolean
character	char
integer	int
real	double
string	String
OPERADORES	
+ - * / %	+ - * / %
== != < <= > >=	== != < <= > >=
not and or	! && ||
CONSTANTES	
true	true
false	false
pi	Math.PI
FUNÇÕES PREDEFINIDAS	
power(<a>, <b>)	Math.pow(a, b)
random(<a>, <b>)	(int)((-a + b + 1) * Math.random() + a)
COMANDOS	
declare <type> <var>	<type> <var>;
<var> = <value>	<var> = <value>;
read <var>	<var> = s.realLine();
print <var> , <value>	System.out.print (<var> + <value);
println <var> , <value>	System.out.println (<var> + <value);
if <cond> then ... else ...	if (<cond>) { ... } else { ... }
while <cond> ...	while (<cond>) { ... }
for <var> from <a> up to <b> step <c>	for (<var>=<a>; <var><=<b>; <var>=<var>+<c>) { ... }
for <var> from <a> down to <b> step <c>	for (<var>=<a>; <var><=<b>; <var>=<var>+<c>) { ... }

Fonte: Elaborado pelo autor.

Da mesma forma que é feito para o enunciado, devem ser demarcados no texto os locais onde serão inseridos os valores aleatórios, através da utilização da sintaxe descrita anteriormente. Além desta demarcação, na seção do pseudocódigo é necessário que sejam inseridos quatro espaços no início de cada linha. Esta adição de espaços se faz necessária para que o pseudocódigo seja exibido com uma fonte diferenciada e com estrutura de linguagem de programação, já que a sintaxe Markdown formata desta maneira quando este número de espaços é adicionado no começo da linha.

É importante salientar que os valores sorteados são utilizados igualmente durante o processo de preenchimento do *template* em questão, ou seja, um parâmetro inteiro "**n**" que foi

sorteado com o valor "**2**" será utilizado para todo o exercício. Com isso, a solução preenchida será equivalente aos valores sorteados no enunciado, mantendo a coerência da atividade com sua solução.

3.2.6 Resultados esperados e dicas

Estas duas seções do *template* foram criadas com o intuito de auxiliar o aluno no entendimento do problema, através do fornecimento de dicas juntamente com os resultados esperados, que são as saídas exibidas pelo exercício, dadas as entradas descritas.

Na etapa da criação do *template*, as dicas e os resultados esperados devem ser adicionados manualmente pelo criador do arquivo. Para delimitação destas seções, os caracteres "##" devem ser utilizados, seguidos de um espaço e o nome em inglês, que neste caso devem ser *hint* e *expected results*. Dentro das duas seções, devem ser inseridas, em apenas uma linha, cada um dos resultados esperados e das dicas, precedidos pelo caractere "-" e um espaço. Caso seja necessário que seja contido algum valor aleatório, da mesma forma que na descrição e na solução, deve ser utilizada a mesma sintaxe "**${**", a variável e concluir com "**}**".

Além das dicas adicionadas manualmente, o sistema pode gerar outras baseadas no pseudocódigo. Esta geração automática será comentada posteriormente, no capítulo 3.3.

3.3 Tradutor

O tradutor é o módulo responsável por receber o pseudocódigo e traduzi-lo para uma linguagem de programação específica. No sistema desenvolvido, é possível gerar até quatro linguagens de programação, sendo elas o Java, C, C++ e Python, ainda possibilitando gerar o pseudocódigo traduzido para o português. Além disso, o módulo permite a criação de dicas geradas automaticamente com base no pseudocódigo. Neste caso, quando o exercício é gerado, além das dicas que já estavam contidas no *template*, outras também são adicionadas na saída.

Para o desenvolvimento deste módulo, foi utilizada a ferramenta ANTLR, que segundo o site ANTLR (2015) é uma ferramenta capaz de gerar analisadores de texto. Este analisador, também conhecido como *parser*, pode ser utilizado para ler, processar, executar ou traduzir um texto estruturado. Através de uma linguagem formal, igualmente chamada de gramática, a ferramenta ANTLR gera automaticamente um analisador de textos, construído na

linguagem Java. Depois de gerado, este arquivo pode receber e analisar qualquer arquivo que siga as regras da gramática especificada na criação do *parser*.

Para criação do analisador, deve ser efetuada uma análise léxica e sintática. Segundo Santiago e Dazzi (2004), a análise léxica é a primeira etapa realizada por um analisador, e sua função é realizar a leitura do texto, caractere por caractere, e transformá-los em símbolos léxicos, conhecidos como *tokens*. Esta lista de símbolos é repassada para a análise sintática, que por sua vez, verifica se toda a cadeia de *tokens* está enquadrada nas regras especificadas. Desta forma, para a criação da gramática, regras léxicas e sintáticas devem ser especificadas para a ferramenta ANTLR.

3.3.1 Criação do arquivo de gramática

O uso da ferramenta inicia com a criação do arquivo de gramática, contendo as regras léxicas e sintáticas. Para o desenvolvimento da gramática do módulo tradutor, foi tomado como base o pseudocódigo desenvolvido. Foram analisados quais os símbolos que devem ser interpretados pela ferramenta, e qual a gramática que deve ser seguida. Na Figura 13 estão detalhadas as regras da gramática proposta para o tradutor, incluindo os símbolos utilizados na análise léxica além das expressões utilizadas na análise sintática.

Figura 13 - Gramática utilizada para o módulo tradutor

```
program → ( statement )*
statement → declare | attribution | read | print | while | if | for
declare → declare type variable newline
attribution → variable = expression newline
read → read variable newline
print → ( print | println ) expression ( , expression )* newline
if → if expression newline indent ( statement ) * dedent ( else newline indent
     ( statement )* dedent )?
while → while expression newline indent ( statement )* dedent
for → for variable from expression ( up | down ) to expression step expression newline
      indent ( statement )* newline
expression → unary_not ( && | || )*
unary_not → ( ! )? exp_comparisson
exp_comparison → exp_arithmetic  ( ( == | != | > | >= | < | <= ) exp_arithmetic )?
exp_arithmetic  → term ( ( + | - ) term )*
term → unary ( ( * | / | % ) unary )*
unary → ( + )? ( - )? factor
factor → number | variable | string | true | false | pi | power ( expression , expression ) |
         random  ( expression , expression ) | ( expression )
type → integer | real | character | boolean | string
```

Fonte: Elaborado pelo autor.

A ferramenta ANTLR permite inserir comandos da linguagem Java junto às regras da gramática. Esses comandos podem receber os *tokens* lidos pelo *parser* e processar os dados conforme a necessidade.

Para o desenvolvimento da tradução do pseudocódigo, foi criada uma variável de controle que armazena a linguagem que deverá ser gerada como saída. Também foram criadas funções que são chamadas conforme as regras são localizadas. Estas recebem os *tokens* lidos pelo *parser* e, conforme o *token* lido e a linguagem definida na variável, armazenam a saída adequada em uma lista de variáveis do tipo texto. Quando todo o texto for lido, a solução estará armazenada nesta lista, bastando buscar todos os dados contidos nela.

Em algumas linguagens, como o Java e o C, a solução deve conter outras informações, como importações no cabeçalho e variáveis globais. Para isso, diversas variáveis do tipo verdadeiro e falso foram criadas, além de listas para serem gravados os cabeçalhos e rodapés das soluções. Conforme as funções são chamadas, estas variáveis são marcadas como verdadeiras. No caso do comando *read* por exemplo, a saída para o Java deve criar uma importação para a classe *scanner*. Nesta situação, quando a regra gramatical localiza este comando no pseudocódigo, a função que gera a saída correta também marca a variável correspondente como verdadeira.

Posteriormente à leitura do arquivo, o sistema analisa a lista com todas as variáveis e identifica quais delas estão marcadas como verdadeiras. Se a linguagem de saída necessitar de algum texto adicional, o mesmo também será adicionado à solução. Para concluir, as listas dos cabeçalhos e rodapés são preenchidas, e por fim, todas são unidas com a solução final.

Para a geração de dicas, foi utilizado o mesmo princípio da escolha da linguagem. Uma variável define o nível das dicas que devem ser geradas. Existe a possibilidade de três níveis. O primeiro é o básico, que gera desde dicas básicas para alunos iniciantes até dicas avançadas. O nível médio mostra dicas úteis para alunos que já estão em uma fase intermediária. E por último, o nível avançado, que apenas mostra dicas mais complexas, para estudantes que já estão mais avançados na área de programação.

O processo de geração é simples, baseado também em variáveis verdadeiro e falso, e nas funções de cada regra gramatical. Para exemplificar, quando uma regra de declaração de uma variável booleana é localizada, será gerada a seguinte dica: “Uma variável booleana deve ser declarada”. Para isso, a variável correspondente é marcada como verdadeira, e no fim de todo o processo de leitura, o sistema analisa quais variáveis são verdadeiras, e qual o nível de dicas deve ser exibido, e adiciona aquelas que se encaixam nos critérios.

3.3.2 Criação das classes *parser* e *lexer*

A ferramenta ANTLR gera duas classes Java como saída. A primeira é a classe léxica, que contém as regras léxicas e os *tokens* da gramática. A segunda é a classe *parser*, ou seja, o analisador. Este contém em sua estrutura, um método *main* que recebe como parâmetro um arquivo contendo o texto a ser analisado.

Para a criação destas classes, é necessário executar o arquivo de gramática juntamente com a biblioteca ANTLR. Para tal, após a criação do arquivo de gramática, o mesmo foi executado e compilado utilizando a ferramenta ANTLR. A Figura 14 demonstra os passos utilizados para a geração dos dois arquivos, onde o "Translator.g" é o arquivo de gramática gerado anteriormente.

Figura 14 - Comandos utilizados para gerar os arquivos do módulo tradutor

```
java -jar antlr-3.2.jar Translator.g

javac -cp antlr-3.2.jar *java
```

Fonte: Elaborado pelo autor.

3.3.3 Alterações finais nos arquivos

Após a criação dos dois arquivos, algumas alterações foram feitas na classe *parser* para possibilitar o uso destas classes em outros programas Java. Primeiramente, como a classe gerada pelo ANTLR é do tipo estática, foi criado um método que reinicia todas os valores das variáveis e das listas utilizadas para a configuração inicial.

Posteriormente, foram adicionadas funções públicas que alteram as variáveis da linguagem de saída e do nível de dicas. A função *main* foi retirada, e em seu lugar, foi criado um método que recebe uma variável do tipo texto, e faz a sua tradução. Por ser uma classe estática, esta última função faz uso do método que reinicia as variáveis toda vez que é acionado. Por fim, foram criados métodos que retornam as listas que contém a solução e as dicas.

Após estas alterações, foi possível utilizar as duas classes geradas juntamente com o módulo gerador, que será detalhado no capitulo 3.4.

3.4 Gerador

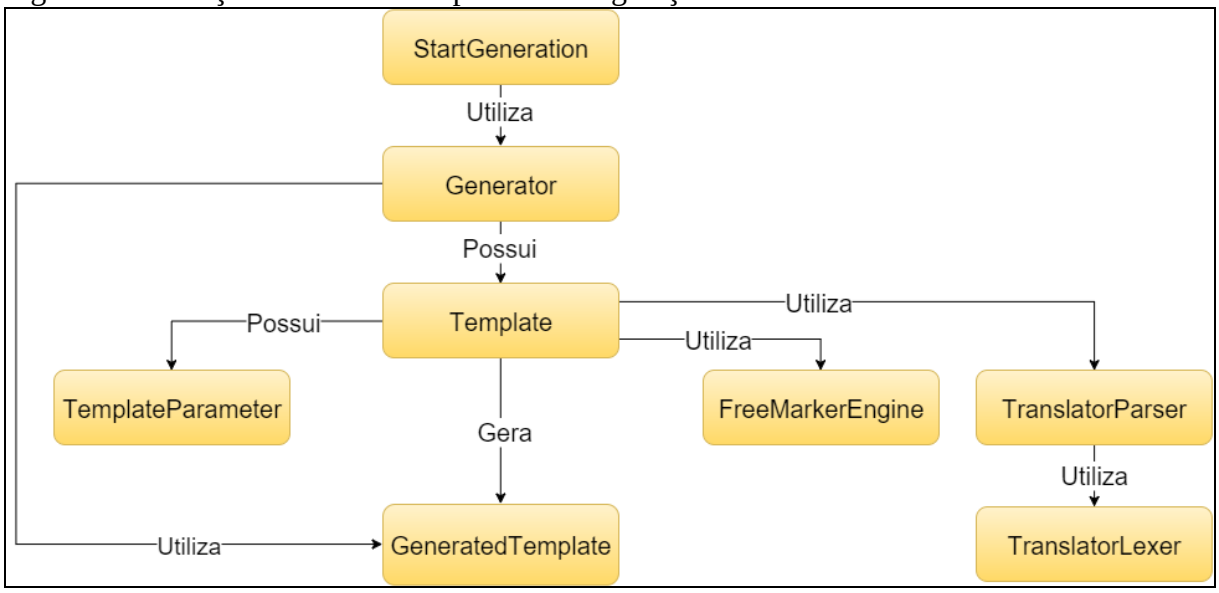
O gerador é responsável por fazer a leitura e o preenchimento dos *templates*, e grava-los formatados de maneira adequada. Além disso, realiza a integração entre os módulos, já que faz uso dos *templates* e do tradutor para gerar os exercícios.

Para o desenvolvimento deste módulo, foi utilizada a linguagem de programação Java juntamente com a ferramenta FreeMarker, que segundo o *site* FREEMARKER (2015) é uma biblioteca utilizada para gerar arquivos com base em *templates*. Esta ferramenta recebe um arquivo de texto com locais demarcados, e uma lista com variáveis e seus respectivos valores. A partir destes dados, cria um novo arquivo com os valores da lista inseridos nos locais especificados pela sintaxe "**${**", seguida do nome da variável e finalizada com "**}**".

O gerador foi desenvolvido através da criação de diversas classes, onde cada uma é um objeto e tem uma função específica para gerar o exercício adequadamente. O módulo é composto pelas seguintes classes: StartGeneration, Generator, Template, TemplateParameter, FreeMarkerEngine, TranslatorParser, TranslatorLexer e GeneratedTemplate.

A classe Generator possui uma lista de objetos Template, que por sua vez, possui dentre seus atributos, uma lista de classes TemplateParameter. O próprio Template faz a geração de um exercício baseado em seu texto. Para isso, faz uso do FreeMarkerEngine, além das classes TranslatorParser e TranslatorLexer criadas anteriormente no módulo tradutor. Como saída, a classe Template gera um objeto do tipo GeneratedParameter. A Figura 15 demonstra esse processo, sendo que cada classe está detalhada no Apêndice A.

Figura 15 - Relação das classes no processo de geração de exercícios



Fonte: Elaborado pelo autor.

A implementação das classes e o processo de geração serão descritos nos tópicos seguintes.

3.4.1 FreeMarkerEngine

Esta classe é utilizada para realizar o preenchimento das lacunas no *template*, e recebe como parâmetros o texto a ser preenchido e um dicionário contendo as variáveis e seus respectivos valores. O FreeMarkerEngine contém a biblioteca FreeMarker e as configurações necessárias para seu funcionamento.

Sempre que um novo objeto deste tipo é instanciado, ocorre a configuração da ferramenta, através da função **configureFreeMarker**. A Listagem 1 demonstra como foi efetuada a configuração.

Listagem 1 – Função que configura a ferramenta Freemarker

```
1       private void configureFreeMarker () {
2               cfg = new Configuration(Configuration.VERSION_2_3_22);
3               cfg.setDefaultEncoding("UTF-8");
4               cfg.setTemplateExceptionHandler(
                  TemplateExceptionHandler.RETHROW_HANDLER);
5       }
```

Fonte: Elaborado pelo autor.

Para realizar o preenchimento das lacunas, foi criado um método público, que recebe o texto (**templateToBeFilled**) e o dicionário de variáveis (**mapParameters**), contendo o nome da variável e seu valor. Estas duas variáveis são enviadas à ferramenta, que retorna uma variável texto com as lacunas preenchidas. A Listagem 1 demonstra como foi criado o método que utiliza a ferramenta FreeMarker para realizar o preenchimento.

Listagem 2 – Procedimento que utiliza o Freemarker para preenchimento de *templates*

```
1       public String fillTemplate (
                String templateToBeFilled,
                Map <String,String> mapParameters)
                throws IOException, TemplateException {
2           freemarker.template.Template temp =
                  new freemarker.template.Template("",
                  new StringReader(templateToBeFilled), cfg);
3           Writer filledTemplate = new StringWriter();
4           temp.process(mapParameters, filledTemplate);
5           return filledTemplate.toString();
6       }
```

Fonte: Elaborado pelo autor.

Para fazer uso desta classe, basta instanciar um novo objeto, e posteriormente utilizar seu método público **fillTemplate**, passando os parâmetros necessários.

3.4.2 TranslatorParser e TranslatorLexer

Estas são as duas classes obtidas através do desenvolvimento do módulo tradutor. Como as duas classes são estáticas, é necessário apenas utilizar os métodos públicos do TranslatorParser que definem a linguagem de saída e o nível de dicas. Com isso pronto, é utilizada a função **translate**, que recebe como parâmetro o texto do pseudocódigo.

Quando finalizado o processo, os métodos **getSolution** e **getHints** são utilizados para retornar respectivamente a solução e as dicas.

3.4.3 TemplateParameter

Esta classe é a representação de cada parâmetro lido de um *template*. É utilizado dentro da classe Template e recebe como parâmetro uma linha da seção *parameters* contida no *template*. Sua função é analisar o trecho de texto recebido e separa-lo, distinguindo o nome e o tipo da variável, juntamente com seu intervalo de dados.

Caso o texto recebido pela classe seja **<n = fixed integer [2 – 6]>**, o nome da variável será a letra **n**, seu tipo será inteiro e o intervalo será entre dois e seis.

Também foi desenvolvida uma função que retorna um valor aleatório que está dentro do intervalo, independentemente do tipo do parâmetro. Desta forma, basta utilizar o método **getRandomValue** para que a classe faça o sorteio do valor deste parâmetro.

3.4.4 Template

A classe Template contém todas as informações de cada um dos *templates* lidos pelo sistema e recebe como parâmetro um objeto contendo o arquivo do *template*. Quando instanciado, este objeto efetua a leitura de todo o texto e separa cada seção. A separação destes trechos fica armazenada em dois grupos de variáveis: as que mantém o texto com a sintaxe Markdown e as que possuem somente a informação necessária, ou seja, sem os caracteres de formatação.

Para fazer a separação de cada seção, foi desenvolvido um método que efetua a leitura de toda a extensão do texto, sempre buscando pelos caracteres “**#**” além do nome da seção. Primeiramente, a função busca o título do enunciado e grava na respectiva variável. Após este

processo, é efetuada a leitura do texto até que a próxima seção seja encontrada, realizando este procedimento até o fim do arquivo.

Por exemplo, para a localização do código do *template* (code), a classe faz a leitura do texto até que encontre o nível (level), que por sua vez, é lido até que seja encontrado o tópico (topic). Sempre que a seção seguinte é encontrada, o conteúdo atual é armazenado na respectiva variável e removido do texto anterior, repetindo o processo até o fim do arquivo. Esta primeira etapa salva todos os dados nas variáveis que contém a sintaxe Markdown, e assim, mantém toda a formatação do texto.

Com estas informações armazenadas, o texto Markdown é retirado. Todos os caracteres da sintaxe Markdown e o nome das seções são removidos, mantendo somente as informações importantes no segundo grupo de variáveis.

Após instanciado, o objeto possui todas as informações do *template*. Estas informações podem ser obtidas pelos diversos métodos que retornam os valores do objeto.

A geração de um exercício é realizada pela própria classe Template, através da função pública **generateOutput**. Para realizar este processo, a classe faz uso de sua lista de TemplateParameters, utilizando a função **generateRandomValue**. Primeiramente, passa por toda esta lista de parâmetros, obtendo um valor aleatório para cada um dos itens. Posteriormente, estes valores são adicionados em um dicionário, e são enviados juntamente com o enunciado e pseudocódigo para a classe FreeMarkerEngine, onde é efetuado o preenchimento de ambos com base nas variáveis do dicionário. Além disso, é verificado se alguma das dicas ou dos resultados esperados possuem campos para serem preenchidos. Caso seja necessário, cada item é enviado também para o FreeMarkerEngine. A Figura 16 mostra um texto com as lacunas e o respectivo retorno da classe, sendo que o parâmetro do *template* era uma variável **n** com intervalo de **2** até **6**.

Figura 16 - Comparativo entre texto com lacunas e texto preenchido

Texto com lacunas

```
## Description

Write a program that asks a string **s** from
the user and then prints it **${n}** times on
the screen, all in a single line and with no
spaces between.

## Pseudocode

    declare string s
    read s
    declare integer i
    for i from 1 up to ${n} step 1
        print s
    println ""
```

Texto preenchido

```
## Description

Write a program that asks a string **s** from
the user and then prints it **5** times on
the screen, all in a single line and with no
spaces between.

## Pseudocode

    declare string s
    read s
    declare integer i
    for i from 1 up to 5 step 1
        print s
    println ""
```

Fonte: Elaborado pelo autor.

Com os dados preenchidos, é feito o envio do pseudocódigo para a classe TranslatorParser, que por sua vez, gera a solução na linguagem solicitada. A Figura 17 mostra o comparativo entre o pseudocódigo anterior já preenchido, com a solução gerada pelo tradutor.

Figura 17 - Comparativo entre o pseudocódigo e a solução gerada

Pseudocódigo preenchido

```
declare string s
read s
declare integer i
for i from 1 up to 5 step 1
    print s
println ""
```

Solução traduzida para Java

```
public class Exercise
{
    public static Scanner scanner = new Scanner (System.in);
    public static void main (String[] args)
    {
        String s;
        s = scanner.nextLine();
        int i;
        for (i=0;i<=5;i=i+1)
        {
            System.out.print(s);
        }
        System.out.println("");
    }
}
```

Fonte: Elaborado pelo autor.

Para finalizar, a classe reúne todas as informações obtidas, e gera o exercício final. Este exercício é armazenado em um objeto do tipo GeneratedTemplate.

3.4.5 GeneratedTemplate

Esta classe é gerada pela classe Template e possui cada uma das seções do *template* formatadas com a linguagem Markdown. Possui métodos que recebem somente o texto (sem formatação) de cada uma das seções. Estes métodos adicionam a formatação necessária para cada uma das partes do exercício, e armazenam nas variáveis da classe.

A classe GeneratedTemplate também possui métodos que retornam cada uma das partes separadamente com o texto formatado. Estas partes são usadas posteriormente para a montagem do exercício no arquivo de saída.

3.4.6 Generator

A classe Generator é responsável por realizar a leitura dos *templates*, montar os exercícios, e armazenar os arquivos de saída em local adequado. Para este processo, faz uso das classes Template e GeneratedTemplate.

Para realizar a leitura dos arquivos, foram desenvolvidos métodos que buscam os arquivos que estão em disco e, para cada um deles, instanciam um objeto do tipo Template, passando o arquivo lido como parâmetro.

No caso da geração dos exercícios, duas funções principais foram desenvolvidas, e cada uma destas, têm uma saída diferente. A primeira tem o nome de **generateExerciseList**, e gera uma lista de exercícios. A segunda gera uma lista de exames, criando os enunciados e as soluções em arquivos separados, com uma estrutura diferenciada do exercício. Esta última tem o nome de **generateTestList**.

A função **generateExerciseList** recebe como parâmetros uma lista contendo o tópico e as *tags*, o nível de dificuldade, a quantidade de exercícios, o caminho onde devem ser gerados os arquivos, a linguagem da solução e o nível das dicas. Inicialmente, ocorre a seleção dos *templates* pelos critérios do usuário. A seleção é feita pela seguinte ordem: nível de dificuldade, tópico principal e por último as *tags*.

Depois do processo de seleção, é feito o sorteio de um dos *templates* selecionados, e com ele, o Generator usa a função **generateOutput**, recebendo como retorno o exercício gerado em um objeto da classe GeneratedTemplate. Com este objeto em memória, utiliza seus métodos de retorno, e monta o exercício conforme a necessidade. Por fim, salva o arquivo montado no local informado.

Os processos de escolha do *template* bem como sua montagem são efetuados diversas vezes, até que a quantidade de exercícios informada seja alcançada.

O método **generateTestList** funciona de forma semelhante à função especificada anteriormente. Porém, ao invés de receber somente a lista com o tópico e as *tags*, recebe um dicionário contendo especificamente cada um dos exercícios do exame, juntamente com a respectiva lista de valores.

A seleção dos *templates* neste caso ocorre para cada exercício do exame, já que os tópicos mudam para cada atividade. Após a seleção, é feita a escolha aleatória e a montagem do exercício. Neste caso, o Generator monta o arquivo de forma diferente, sendo que um deles contém somente os títulos e os enunciados, e no outro, o pseudocódigo, as dicas e a solução. Os dois arquivos também são gerados formatados para Markdown.

3.4.7 StartGeneration

Esta é a primeira classe utilizada pelo sistema, e tem a função de receber os parâmetros enviados por linha de comando, separa-los de forma adequada e iniciar a geração dos exercícios, através de um objeto Generator.

Os parâmetros que o usuário insere no sistema por linha de comando são armazenados em um vetor. Inicialmente, a classe faz a separação dos dados contidos neste vetor, identificando cada um dos valores, e armazenando em variáveis. Caso algum dos parâmetros esteja incorreto, um erro é exibido, e o sistema é encerrado.

Com as variáveis preenchidas, a classe identifica qual o tipo de saída foi solicitada pelo usuário. Possuindo esta informação, instancia um objeto do tipo Generator, e envia os dados lidos para a função **generateExerciseList**, para uma lista de exercícios, ou para a função **generateTestList**, para a lista de testes.

3.4.8 Processo de geração dos exercícios

O sistema inicia o funcionamento utilizando a classe StartGeneration, que recebe os dados do usuário por linha de comando, separa as informações e envia à classe Generator, que por sua vez, faz a leitura dos *templates* e instancia um objeto Template para cada arquivo, armazenando-os em uma lista.

Após este processo, a classe StartGeneration passa ao Generator qual exercício de saída será gerado. Com isso, o Generator inicia o processo de geração, separando os *templates* pelos critérios. Posteriormente, sorteia um dos objetos Template selecionados e solicita um exercício gerado. Como resposta, terá um objeto GeneratedTemplate, e com ele, faz a

montagem do arquivo de saída e salva no local indicado. Este processo é repetido até que o número de exercícios informado pelo usuário seja alcançado.

Quando todos os arquivos são gerados, o sistema encerra seus processos.

4 RESULTADOS OBTIDOS

Este capítulo tem por objetivo descrever os resultados obtidos através do desenvolvimento do sistema. Inicialmente, será demonstrado o funcionamento, e em seguida, os casos de uso fornecidos pelo software criado.

4.1 Funcionamento do sistema

Além de ser utilizado por professores que almejam criar listas de exercícios e testes, o sistema desenvolvido também pode ser utilizado por ambientes *online* de autoestudo de programação, com intuito de diversificação das listas de exercícios. Por este motivo, o sistema não possui uma interface gráfica e funciona através de linha de comando, sendo necessário ser inicializado por um interpretador de comandos juntamente com o Java Runtime Environment (JRE).

O arquivo executável do sistema é um Java Archive (JAR) com o nome de Generator. Para executá-lo é necessário utilizar a sintaxe do Java, através do comando **java –jar Generator.jar**.

Além disso, posteriormente ao comando de inicialização, devem ser inseridos os parâmetros para o sistema. Estes são os dados de especificação do usuário, como por exemplo o número de exercícios, a linguagem de saída e as *tags*. A Figura 18 mostra um comando completo para utilização do sistema.

Figura 18 - Exemplo de comando para utilização do sistema

```
1 >> java -jar Generator.jar -t exerciselist -lv simple
2    -i string math -n 15 -ln python -h basic -o C:\Exercise
```

Fonte: Elaborado pelo autor.

Cada um dos parâmetros deve ser iniciado pelo caractere "**-**" juntamente com o respectivo nome. Posteriormente deve ser adicionada a opção desejada precedida de um espaço. As opções de parâmetro são: tipos de saída, rótulos do exercício, número de exercícios, linguagem de saída, caminho de saída, nível das dicas e o nível de dificuldade. Cada parâmetro será detalhado nos tópicos a seguir.

4.1.1 Tipos de saída

O tipos de saída são definidos pelo parâmetro "**-t**", e as opções são **exerciselist** e **testlist**, sendo que a primeira gera uma lista de exercícios simples, onde ocorre a geração de um arquivo para cada exercício. Já a segunda opção, tem como saída uma lista de exames, onde, para cada exercício, existe um arquivo contendo os enunciados e no outro, as respostas e as dicas.

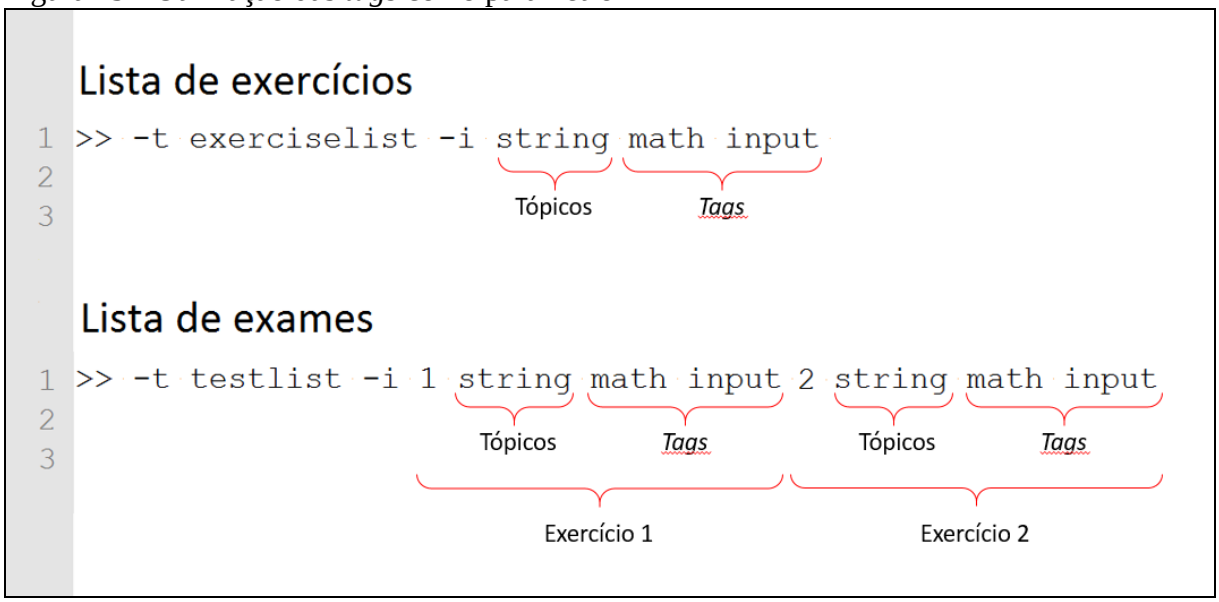
4.1.2 Rótulos do exercício

Os rótulos do exercício são compostos pelo tópico e *tags* de seleção e têm como parâmetro o "**-i**". Estes são os assuntos que serão filtrados pelo gerador, e devem ser inseridos sempre com um espaço entre cada valor. Importante salientar que o sistema considera o primeiro valor inserido como tópico, e os demais como as outras *tags* para seleção. Neste parâmetro, somente o tópico é obrigatório, ou seja, o primeiro valor sempre deve existir, entretanto, as *tags* não são exigidas.

O gerador busca pelos valores contidos nas seções *topic* e *tags* do *template*. Para o filtro, inicialmente é efetuada a busca por todos os arquivos que possuem o tópico inserido. Quando a busca é finalizada, o sistema verifica se existem *tags* no parâmetro, e caso positivo, efetua um novo filtro, a partir dos exercícios selecionados pelo tópico.

As duas possíveis saídas do sistema são uma lista de exercícios e uma lista de exames. Nos exames, cada arquivo gerado contém vários exercícios, como uma prova aplicada em sala de aula, onde cada teste tem diversos exercícios. Por isso, este parâmetro tem uma diferença dependendo do tipo de saída. Caso o tipo escolhido for o **exerciselist**, é necessário colocar apenas uma vez os valores, sendo eles separados por espaço. Se o tipo de saída escolhido for o **testlist**, estes valores devem ser inseridos para cada exercício do exame, sendo necessário adicionar antes de cada lista de dados, o número do exercício. A Figura 19 demonstra a utilização deste parâmetro, destacando a diferença entre as duas entradas.

Figura 19 - Utilização das *tags* como parâmetro



Fonte: Elaborado pelo autor.

4.1.3 Número de exercícios

O número de exercícios é definido pelo parâmetro "**-n**" e define a quantidade de exercícios que serão gerados, podendo ser qualquer número maior do que zero. É importante salientar que se o tipo de exercício está definido como lista de exercícios, a quantidade de arquivos gerados será exatamente igual ao número do parâmetro. Porém, se a opção lista de exames for utilizada, o número de arquivos será duas vezes maior, já que para cada exercício, são gerados dois arquivos.

4.1.4 Linguagem de saída

O parâmetro da linguagem de saída é definido pelos caracteres "**-ln**". Conforme já mencionado, o sistema possibilita gerar soluções em até quatro linguagens de programação, sendo elas Java, C, C++ e Python, ainda possibilitando gerar a saída traduzida do pseudocódigo. As opções deste parâmetro são respectivamente **java**, **c**, **c++**, **python** e **portuguese**.

4.1.5 Caminho de saída

Este parâmetro recebe o caminho em que os arquivos gerados devem ser salvos. É definido pelos caracteres "**-o**". Sua utilização é simples, bastando informar um caminho válido de saída para que os arquivos sejam salvos.

Caso o caminho inserido esteja inválido, o sistema não cria as pastas necessárias até o local onde serão salvos os arquivos, e assim, apenas mostra um erro informando que o caminho especificado não existe.

4.1.6 Nível das dicas

Este parâmetro define qual nível de dicas deverá ser gerado pelo módulo tradutor. É definido pelos caracteres "**-h**", e recebe como opções dicas básicas, médias e avançadas, sendo determinadas respectivamente pelos termos **basic**, **medium** e **advanced**.

Quando solicitado dicas básicas, o sistema exibe todas as dicas possíveis, desde as mais simples, como declarações de variáveis, até as mais avançadas, como uso de termos matemáticos. Este nível de dicas é voltada aos estudantes iniciantes, com pouca ou nenhuma experiência em programação.

As dicas do nível médio são voltadas a estudantes intermediários, que já tem algum conhecimento, e não necessitam de dicas triviais. Sendo assim, o sistema não exibe as dicas mais simples. Já o nível avançado, exibe somente as dicas mais complexas, sendo estas voltadas a alunos já avançados, que já têm um conhecimento maior sobre lógica de programação.

4.1.7 Nível de dificuldade

Este parâmetro filtra os exercícios pelo seu nível de dificuldade, ou seja, pelo valor definido na seção *level* do *template*. As opções para este parâmetro são os exercícios simples, médios ou avançados, sendo nomeados respectivamente pelos termos **simple**, **medium** e **advanced**.

Para este trabalho, somente foram desenvolvidos *templates* de nível simples, devido à complexidade do pseudocódigo só permitir exercícios mais simplificados.

4.2 Casos de uso

Conforme já mencionado anteriormente, o sistema possibilita dois casos de uso, sendo eles uma lista de exercícios ou uma lista de exames. A lista de exercícios é voltada a diversificação das atividades, tanto em ambientes *online*, quanto para uso manual. Já a lista de exames tem como proposito a criação de testes ou provas, e por isso tem uma saída diferenciada, contendo enunciado e respostas em locais separados.

Nos tópicos a seguir, os casos de uso serão detalhados juntamente com a apresentação de exercícios gerados em cada situação.

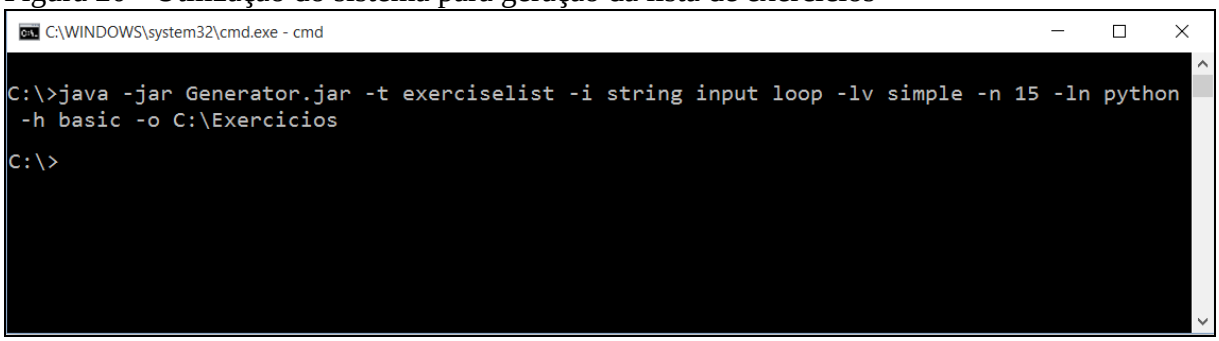
4.2.1 Lista de exercícios

A lista de exercícios gerada pelo parâmetro **exerciselist** tem como saída uma lista de arquivos onde cada um deles contém todos os dados do exercício, ou seja, cada arquivo salvo é equivalente a um exercício, e todas as informações como o enunciado, pseudocódigo, dicas e solução estão contidas nele.

Conforme já comentado, todo o exercício gerado já está formatado com a sintaxe do Markdown, e assim, basta que os trechos necessários sejam copiados do arquivo de saída e disponibilizados em um ambiente que exibe a formatação desta linguagem.

Os exercícios serão gerados com base nos critérios do usuário que foram inseridos através dos parâmetros de entrada. A Figura 20 mostra a utilização do sistema gerador de exercícios, sendo que serão gerados 15 exercícios, com o tópico principal sendo **string**, e contendo no exercício os assuntos **input** e **loop**. Além disso, o nível de dificuldade será simples, com um nível básico de dicas e os arquivos sendo gerados no caminho "C:\Exercicios".

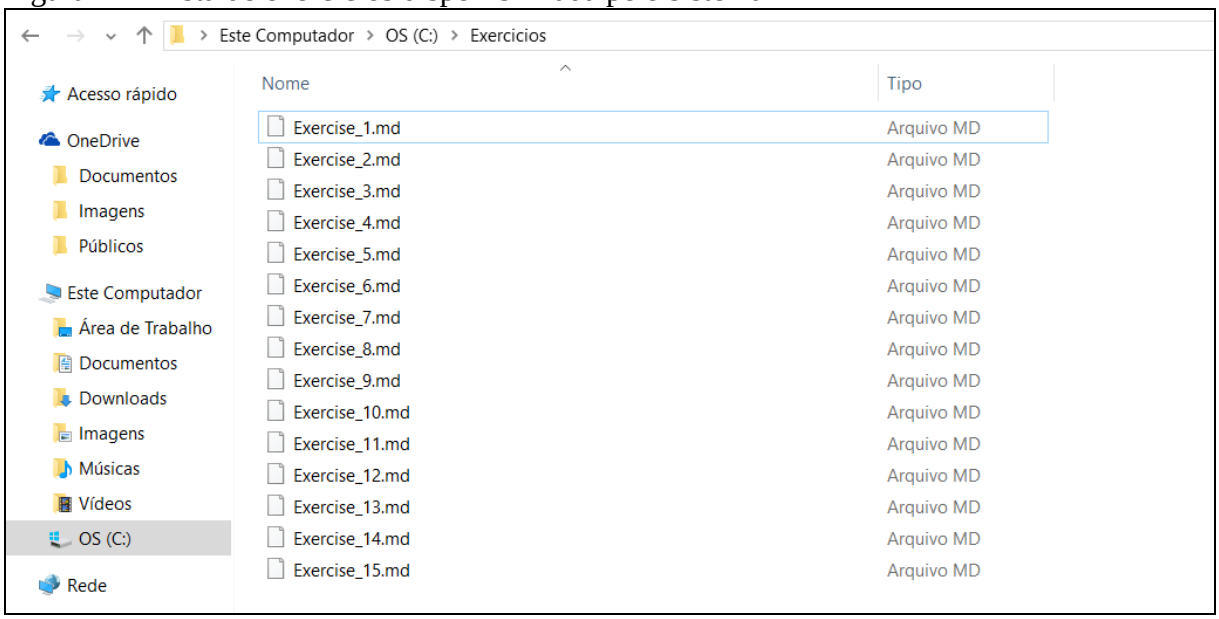
Figura 20 - Utilização do sistema para geração da lista de exercícios



Fonte: Elaborado pelo autor.

Após o processo de geração dos exercícios, a lista de arquivos estará disponível no caminho especificado, conforme demonstrado pela Figura 21.

Figura 21 - Lista de exercícios disponibilizada pelo sistema



Fonte: Elaborado pelo autor.

A lista de exercícios pode conter vários exercícios diferentes, porém, todos os exercícios selecionados e salvos possuem o tópico e as *tags* contidas nos parâmetros inseridos pelo usuário. A Listagem 3 demonstra um dos exercícios gerados.

Listagem 3 – Exercício gerado pelo sistema

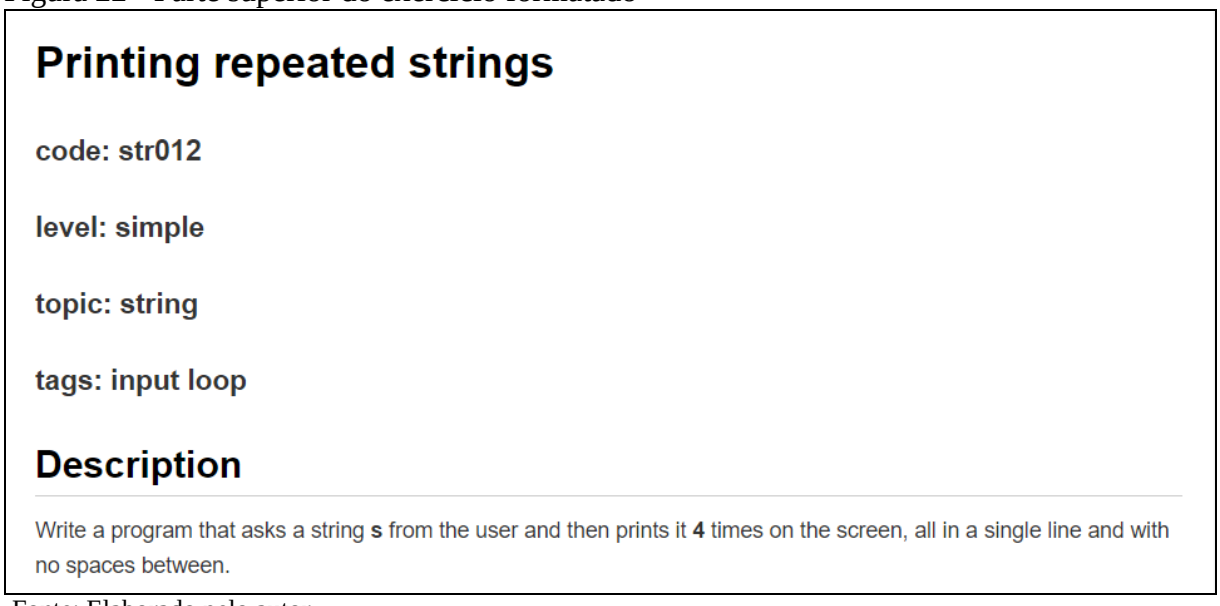
```
1       # Printing repeated strings
2
3       ### code: str012
4       ### level: simple
5       ### topic: string
6       ### tags: input loop
7
8       ## Description
9
10      Write a program that asks a string **s** from
11      the user and then prints it **4** times on the
12      screen, all in a single line and with no spaces
13      between.
14
15      ## Pseudocode
16
17          declare string s
18          read s
19          declare integer i
20          for i from 1 up to 4 step 1
21              print s
22          println ""
23
24      ## Expected Results
25
```

```
26      - An input of **"abc"** should result in: **"abcabcabc"**
27
28      ## Hints
29
30      - Use a repetition, counting from 1 to **4**.
31      - Use the operator **+** for concatenating strings.
32      - The 'Print' statement should be used.
33      - The 'Print Line' statement should be used.
34      - The 'Read' statement should be used.
35
36      ## Solution
37
38          s = input()
39          for i in range (1,4+1,1):
40              print (s, end="")
41          print ("")
42
```

Fonte: Elaborado pelo autor.

Quando este mesmo arquivo gerado é disponibilizado em algum ambiente que aceite a linguagem Markdown, o exercício fica em sua versão formatada. Para uma melhor visualização, o arquivo formatado foi separado em três imagens. A Figura 22 mostra a parte superior, onde ficam situados o título, as informações do exercício e a descrição.

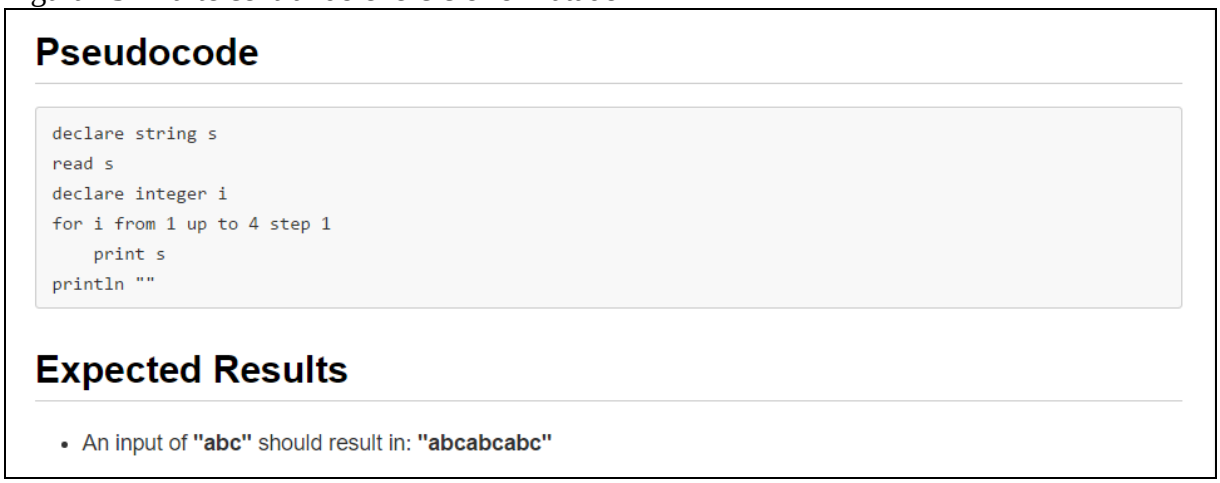
Figura 22 - Parte superior do exercício formatado



Fonte: Elaborado pelo autor.

A parte central contém o pseudocódigo e os resultados esperados, conforme pode ser visto na Figura 23.

Figura 23 - Parte central do exercício formatado



Pseudocode

```
declare string s
read s
declare integer i
for i from 1 up to 4 step 1
    print s
println ""
```

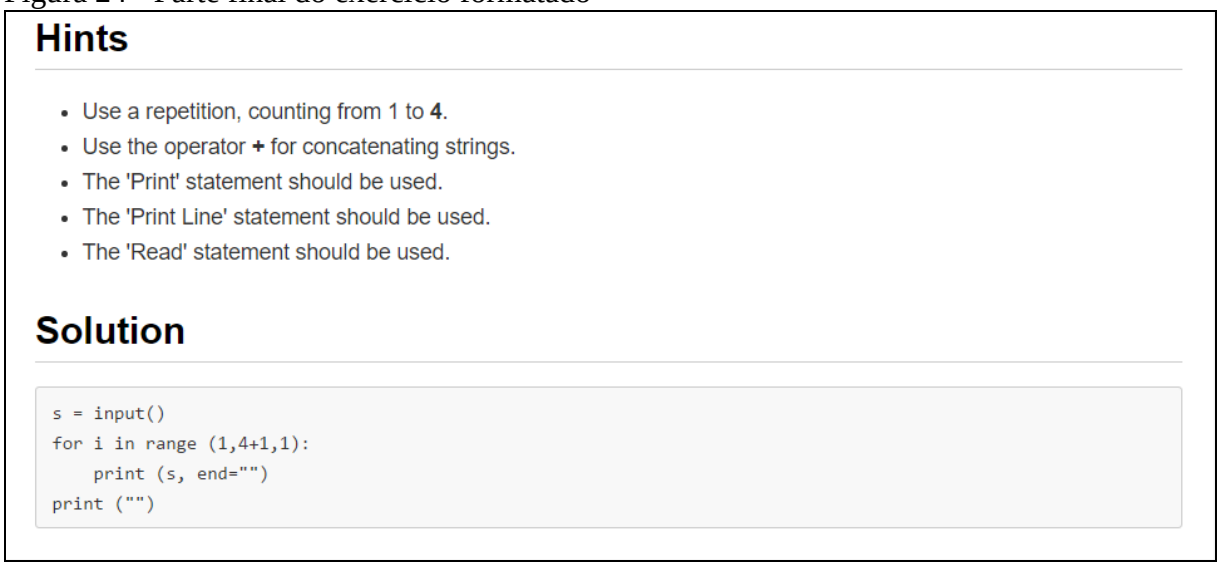
Expected Results

- An input of **"abc"** should result in: **"abcabcabc"**

Fonte: Elaborado pelo autor.

Já a parte final do arquivo contém as dicas e a solução do exercício. Neste caso, conforme os parâmetros inseridos, é possível verificar que a solução gerada está na linguagem de programação Python. A Figura 24 mostra o último trecho do exercício.

Figura 24 - Parte final do exercício formatado



Hints

- Use a repetition, counting from 1 to **4**.
- Use the operator **+** for concatenating strings.
- The 'Print' statement should be used.
- The 'Print Line' statement should be used.
- The 'Read' statement should be used.

Solution

```
s = input()
for i in range (1,4+1,1):
    print (s, end="")
print ("")
```

Fonte: Elaborado pelo autor.

As imagens que contém o texto formatado são a representação do exercício disponibilizado ao aluno, contendo todas as informações. Caso o seja necessário, é possível recortar do texto gerado apenas as partes necessárias. Assim, só será exibido as partes que mais importam no momento da atividade.

4.2.1.1 Compilação da solução gerada

Para fins de teste, a solução gerada pelo sistema foi testada e analisada para verificar se está funcionando corretamente. Com isso, o código fonte gerado foi compilado através do site Ideone[2]. A Figura 25 retrata o teste efetuado, onde os valores de entrada foram os caracteres “abc”.

Figura 25 - Teste efetuado compilando a saída fornecida pelo sistema

Fonte: Elaborado pelo autor.

A partir do teste, foi analisado que a solução fornecida pelo sistema é compilável, e com isso, constatou-se que a saída está funcionando corretamente.

4.2.2 Lista de exames

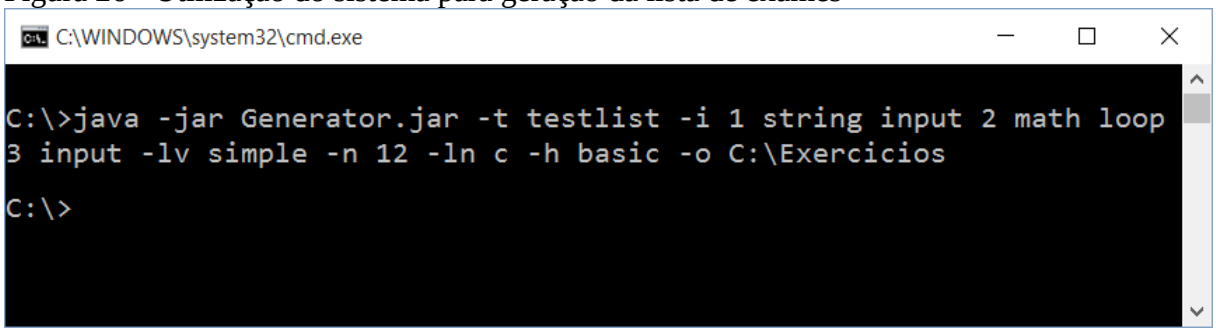
O outro caso de uso do sistema tem como propósito a aplicação de exames ou testes. Esta lista é gerada pelo parâmetro **testlist**, e tem algumas diferenças em relação aos exercícios apresentados anteriormente. No caso das provas, a saída possui dois arquivos por exercício onde um dos arquivos contém as atividades da prova (que será disponibilizada ao aluno), e no outro ficam as dicas, resultados esperados e a solução, ficando a cargo do professor disponibilizar ou não estes dados.

[2] Ideone é um compilador *online* que permite compilar e executar código fonte em mais de 60 linguagens de programação (IDEONE, 2015).

Neste caso, os exercícios também já são gerados formatados pela sintaxe Markdown, com pequenas diferenças, como o título de cada questão, que fica separado por duas listas, para detalhar melhor cada atividade.

Para exemplificar este caso de uso, foi gerada uma lista de 12 exames, sendo que a prova tem três exercícios. O primeiro contém como tópico principal **strings** e o termo **input** como *tag*. Já o segundo exercício contém o tópico como **math** e **loop** e o terceiro somente o tópico como **input**. Os exercícios foram salvos em "C:\Exercicios", sendo que o nível de dicas gerado foi o nível básico, o nível de dificuldade selecionado é simples e a linguagem de saída o C. A Figura 26 demonstra a execução do sistema.

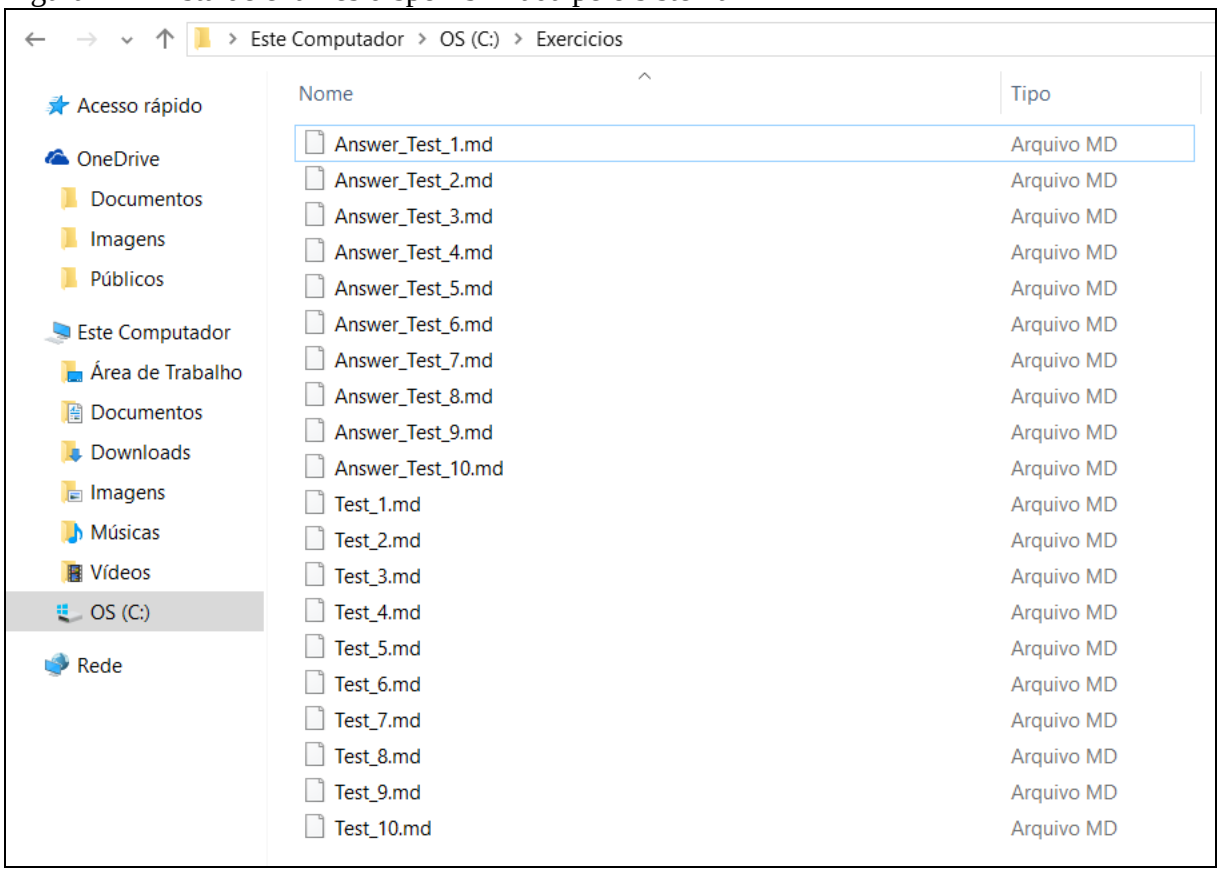
Figura 26 - Utilização do sistema para geração da lista de exames



Fonte: Elaborado pelo autor.

Após a utilização do sistema, a lista de arquivos gerados ficou disponível no caminho informado no parâmetro. A Figura 27 mostra que foram gerados dois arquivos por exercício, onde um deles contém os enunciados, e o outro armazena as respostas.

Figura 27 - Lista de exames disponibilizada pelo sistema



Fonte: Elaborado pelo autor.

A lista exames contém diferentes exercícios em cada prova, mas segue os parâmetros inseridos pelo usuário para cada exercício. A Listagem 4 mostra uma prova gerada pelo sistema.

Listagem 4 – Exame gerado pelo sistema

```
1       ___
2       # Exercise 1 - Printing repeated strings
3       ___
4
5       ## Description
6
7       Write a program that asks a string **s** from the user and then
8       prints it **2** times on the screen, all in a single line and
9       with no spaces between.
10
11      ___
12      # Exercise 2 - Average
13      ___
14
15      ## Description
16
17      Write a program that asks **4** numbers from the user and then
```

```
18      prints the average on the screen.
19
20      ___
21      # Exercise 3 - Bigger or smaller
22      ___
23
24      ## Description
25
26      Write a program that asks a number **n** from the user and then
27      prints "Bigger" if the number is greater than **4** or prints
28      "Smaller" if the number is smaller.
29
```

Fonte: Elaborado pelo autor.

O arquivo de respostas contém os dados de todos os exercícios, fazendo com que seja muito grande. Por isso, a Listagem 5 mostra somente as respostas do segundo exercício da prova anterior, sendo que todas as outras respostas seguem o mesmo formato.

Listagem 5 – Respostas do segundo exercício gerado pelo sistema

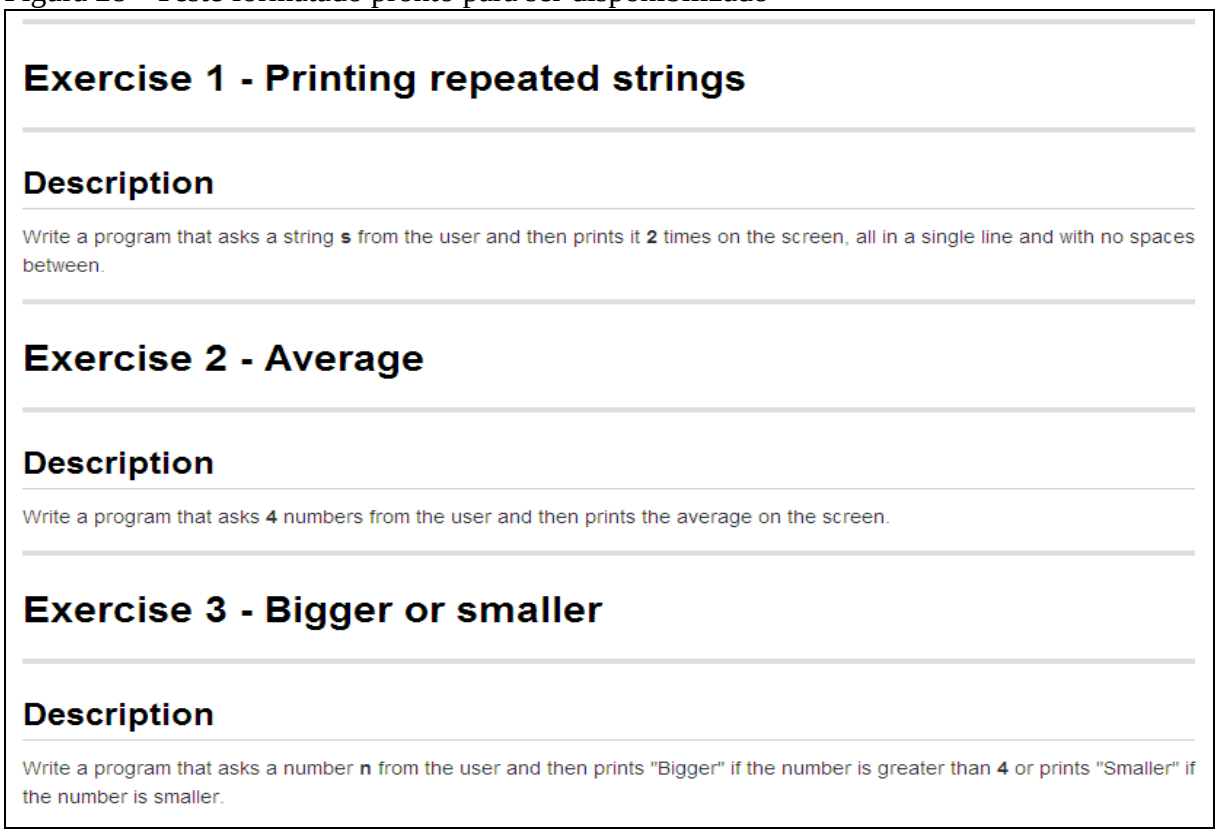
```
1       ___
2       # Exercise  2 - *Hints and Answers*
3       ___
4       # Average
5
6       ## Hints
7
8       - Use a repetition, counting from 1 to **6**.
9       - Read **6** times the number from the user.
10      - An integer should be declared.
11      - A real (double) should be declared.
12      - A 'less or equal' operator should be used.
13      - The 'Print Line' statement should be used.
14      - The 'Read' statement should be used.
15
16      ## Expected Results
17
18      - An input of **"6 7 8"** should result in: **"7"**
19
20      ## Pseudocode
21
22          declare real n
23          declare integer count
24          declare integer avg
25          avg = 0
26          count = 1
27          while count <= 6
28              read n
29              avg = avg + n
30              count = count + 1
31          avg = avg / 6
32          println avg
```

```
33
34      ## Solution
35
36          #include <stdio.h>
37          int main()
38          {
39              double n;
40              int count;
41              double avg;
42              avg = 0;
43              count = 1;
44              while (count <= 6)
45              {
46                  scanf("%lf", &n);
47                  avg = avg+n;
48                  count = count+1;
49              }
50              avg = avg/6;
51              printf("%f \n", avg);
52          }
```

Fonte: Elaborado pelo autor.

O exame também é gerado formatado para a sintaxe Markdown. A Figura 28 mostra o teste formatado, pronto para ser disponibilizado ao aluno.

Figura 28 - Teste formatado pronto para ser disponibilizado



Exercise 1 - Printing repeated strings

Description

Write a program that asks a string **s** from the user and then prints it **2** times on the screen, all in a single line and with no spaces between.

Exercise 2 - Average

Description

Write a program that asks **4** numbers from the user and then prints the average on the screen.

Exercise 3 - Bigger or smaller

Description

Write a program that asks a number **n** from the user and then prints "Bigger" if the number is greater than **4** or prints "Smaller" if the number is smaller.

Fonte: Elaborado pelo autor.

5 CONCLUSÃO

Este trabalho teve como objetivo o desenvolvimento de um sistema gerador de exercícios de programação, com o intuito de auxiliar no ensino das disciplinas de algoritmos e programação. A proposta de desenvolver tal sistema surgiu em virtude de pesquisas realizadas na área de ensino, especificamente na área da Tecnologia da Informação, onde foi constatado que o ensino de algoritmos ainda segue o formato tradicional. Porém, já estão surgindo técnicas e ferramentas *online* que fornecem um ambiente agradável e motivador para alunos que têm necessidade de aprender sobre este assunto. Entretanto, foi verificado que ainda não havia sido desenvolvido um sistema capaz de gerar exercícios de programação automaticamente.

Com isso, o presente trabalho teve como proposta o desenvolvimento desta ferramenta, almejando não só a diversificação de listas de exercícios, mas buscando também formas de auxiliar os alunos no aprendizado, já que existe uma dificuldade em disciplinas de algoritmos. Além disso, buscou-se uma melhora na qualidade e disponibilização das atividades disponibilizadas nos ambientes *online* de autoestudo.

Para o desenvolvimento do sistema, foi elaborado um modelo de *template* que é utilizado como base para a geração dos exercícios. Este *template* contém todas as informações necessárias para a geração de cada atividade. Posteriormente foi criado um pseudocódigo que se assemelha ao Portugol, entretanto, com termos na língua inglesa. Esta linguagem estruturada foi criada para facilitar a disponibilização das soluções dos exercícios, permitindo que fossem geradas saídas em até quatro linguagens de programação juntamente com a tradução do pseudocódigo. Além disso, o pseudocódigo pode ser utilizado pelos alunos para auxiliar no entendimento de cada atividade. Por fim, foi desenvolvido o sistema que seleciona, preenche e salva os arquivos, fornecendo dicas e dados que auxiliam no entendimento do conteúdo.

O sistema desenvolvido proporciona diversas facilidades na área de ensino de programação, como a criação e disponibilização de listas de exercícios e testes para auxiliar no trabalho dos professores, diversificação de atividades em ambientes *online* de autoestudo e fornecimento de informações didáticas aos alunos com intuito de auxiliar no aprendizado.

Além do que já foi desenvolvido, algumas funcionalidades poderiam ser adicionadas no sistema, como a integração com os *online judges*, que permitiriam a validação das respostas dos alunos juntamente com a solução fornecida pelo software. Já o pseudocódigo elaborado poderia ser melhorado, adicionando mais comandos, utilização de funções ou mais tipos de variáveis como vetores e matrizes, permitindo assim, a geração de soluções e exercícios mais complexos. Ainda é possível desenvolver um dicionário de termos, que poderia ser utilizado para buscar valores aleatórios e inseri-los no texto ou na solução, sem ter necessidade de coloca-los no *template*.

REFERÊNCIAS

ALBERTI, T. F. et al. OPORTUNIDADES, PERSPECTIVAS E LIMITAÇÕES DOS MOOC NO ÂMBITO DA UAB/UFSM. In: X Congresso Brasileiro de Ensino Superior a Distância, 2013, Belém. **Anais eletrônicos...** Belém, 2013. Disponível em: <http://www.aedi.ufpa.br/esud/trabalhos/poster/AT1/114256.pdf>. Acesso em: 21 abril 2015.

ALICE. **An educational software that teaches students computer programming in a 3D environment**. Disponível em: <http://www.alice.org/index.php>. Acesso em 12 maio. 2015.

ANTLR. **ANTLR**. Disponível em: <http://www.antlr3.org/>. Acesso em 02 junho 2015.

AURELIANO, V. C. O.; TEDESCO, P. C. de A. R. **Avaliando o uso do Scratch como abordagem alternativa para o processo de ensino-aprendizagem de programação**, 2012. Disponível em: <http://www.lbd.dcc.ufmg.br/colecoes/wei/2012/006.pdf>. Acesso em: 17 maio 2015.

BARIN, C. S.; BASTOS, F. da P. de. Problematização dos MOOC na atualidade: Potencialidades e Desafios. **Revista Renote**. CINTED-UFRGS, Porto Alegre, V. 11, n. 3, dezembro 2013. Disponível em: <http://www.seer.ufrgs.br/index.php/renote/article/view/44707/28546>. Acesso em: 21 abril 2015.

BASTOS, R. C.; BIAGIOTTI, B. MOOCs: uma alternativa para a democratização do ensino. **Revista Renote**. CINTED-UFRGS, Porto Alegre, V. 12, n. 1, julho 2014. Disponível em: <http://www.seer.ufrgs.br/index.php/renote/article/view/50333/31417>. Acesso em: 21 abril 2015.

BORGES, M. A. F. Avaliação de uma metodologia alternativa para a aprendizagem de programação. In: VIII Workshop de Educação em Computação, 2000, Curitiba. **Anais eletrônicos...** Disponível em: <http://www.niee.ufrgs.br/eventos/SBC/2000/pdf/wei/relatos/selecionados/wei006.pdf>. Acesso em: 17 maio 2015.

CAIZA, J. C.; ALAMO, J. M. Del. PROGRAMMING ASSIGNMENTS AUTOMATIC GRADING: REVIEW OF TOOLS AND IMPLEMENTATIONS. In: 7th International Technology, Education and Development Conference, 2013, Valencia. **Proceedings...** Disponível em: <http://oa.upm.es/25765/>. Acesso em: 17 maio 2015.

CHAVES, J. O. et al. MOJO: UMA FERRAMENTA PARA AUXILIAR O PROFESSOR EM DISCIPLINAS DE PROGRAMAÇÃO. In: X Congresso Brasileiro de Ensino Superior a Distância, 2013, Belém. **Anais eletrônicos...** Belém, 2013. Disponível em: <http://www.aedi.ufpa.br/esud/trabalhos/poster/AT3/114160.pdf>. Acesso em: 28 abril 2015.

CODECADEMY**. Learn to code interactively, for free**. Disponível em: **<** http://web-cat.org/group/codeworkout**>. Acesso em 17 maio 2015.**

CODEWORKOUT. **The Web-Cat Community:** CodeWorkout. Disponível em: < http://web-cat.org/group/codeworkout>. Acesso em 17 maio 2015.

DARING FIREBALL. **Markdown**. Disponível em: < https://daringfireball.net/projects/markdown/ >. Acesso em 09 novembro 2015.

DANN, W. et al. Mediated Transfer: Alice 3 to Java. In: 43rd ACM Technical Symposium on Computer Science Education, 2012, New York. **Proceedings...** Disponível em < http://dl.acm.org/citation.cfm?id=2157180>. Acesso em: 17 maio 2015.

FARDO, M. L. A GAMIFICAÇÃO APLICADA EM AMBIENTES DE APRENDIZAGEM. **Revista Renote**. CINTED-UFRGS, Porto Alegre, V. 11, n. 1, julho 2013. Disponível em: <http://www.seer.ufrgs.br/index.php/renote/article/view/41629/26409>. Acesso em: 28 abril 2015.

FERRADIN, M.; STEPHANI, S. L. Ferramenta para o ensino de Programação via Internet. In: SULCOMP, 2005. **Anais eletrônicos...** Disponível em: <http://periodicos.unesc.net/index.php/sulcomp/article/viewArticle/794>. Acesso em: 17 maio 2015.

FREEMARKER. **FreeMarker**. Disponível em: <http://freemarker.org/>. Acesso em 02 junho 2015.

HAMARI, J.; KOIVISTO, J.; SARSA, H. **Does Gamification Work? — A Literature Review of Empirical Studies on Gamification**. IEEE Computer Society, 2014. Disponível em: <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6758978>. Acesso em: 28 abril 2015.

IDEONE. **Ideone**. Disponível em: < https://ideone.com/>. Acesso em 10 novembro 2015.

JÚNIOR, J.C.R.P.; RAPKIEWICZ, C. E. **O Processo de Ensino-Aprendizagem de Fundamentos de Programação: Uma Visão Crítica da Pesquisa no Brasil**. 2004. Disponível em: <http://www.lbd.dcc.ufmg.br/colecoes/weirjes/2004/003.pdf>. Acesso em: 09 maio 2015.

KLOCK, A. C. T. et al. Análise das técnicas de Gamificação em Ambientes Virtuais de Aprendizagem. **Revista Renote**. CINTED-UFRGS, Porto Alegre, V. 12, n. 2, dezembro 2014. Disponível em: <http://www.seer.ufrgs.br/index.php/renote/article/view/53496/33013>. Acesso em: 28 abril 2015.

KHANACADEMY. **Você só precisa saber uma coisa:** Você pode aprender qualquer coisa. Disponível em: <https://pt.khanacademy.org/>. Acesso em 17 maio 2015.

LEARNEROO. **Learn by doing**. Disponível em: < https://www.learneroo.com/>. Acesso em 17 maio 2015.

MATTA, C. E. da; FIGUEIREDO, A. P. S. MOOC: TRANSFORMAÇÃO DAS PRÁTICAS DE APRENDIZAGEM. In: X Congresso Brasileiro de Ensino Superior a Distância, 2013, Belém. **Anais eletrônicos...** Belém, 2013. Disponível em: <http://www.ead.unb.br/arquivos/artigos/mooc_artigo_esud2013.pdf>. Acesso em: 21 abril 2015.

MÉLO. F.É. N. et al. DO SCRATCH AO ARDUINO: UMA PROPOSTA PARA O ENSINO INTRODUTÓRIO DE PROGRAMAÇÃO PARA CURSOS SUPERIORES DE TECNOLOGIA. In: XXXIX Congresso Brasileiro de Educação em Engenharia, 2011, Blumenau. **Anais eletrônicos…** Disponível em: <http://www3.fsa.br/LocalUser/cobenge2011/sessoestec/art1886.pdf>. Acesso em: 17 maio 2015.

IHANTOLA, P. et al. Review of Recent Systems for Automatic Assessment of Programming Assignments. In: 10th Koli Calling International Conference on Computing Education, 2010, New York. **Proceedings…** Disponível em <http://dl.acm.org/citation.cfm?id=1930480>. Acesso em: 17 maio 2015.

RAPKIEWICZ, C. E. et al. ESTRATÉGIAS PEDAGÓGICAS NO ENSINO DE ALGORITMOS E PROGRAMAÇÃO ASSOCIADAS AO USO DE JOGOS EDUCACIONAIS. **Revista Renote**. CINTED-UFRGS, Porto Alegre, V. 4, n. 2, dezembro 2006. Disponível em: <http://www.seer.ufrgs.br/renote/article/view/14284/0>. Acesso em: 09 maio 2015.

ROBOCODE. **Build the best - destroy the rest!**. Disponível em: <http://robocode.sourceforge.net/>. Acesso em 12 maio 2015.

ROQUE, A. S. et al. **Técnicas de Gameficação em AVAs:** Um Estudo de Caso no Ambiente Virtual de Aprendizagem Moodle. In: Encontro Anual de Tecnologia da Informação e Semana Acadêmica de Tecnologia da Informação, 2013. **Anais eletrônicos...** Frederico Westphalen, 2013. Disponível em: <http://www.eati.info/eati/2013/assets/anais/artigo53.pdf >. Acesso em: 05 maio 2015.

SANTIAGO, R; DAZZI, R. L. S. Interpretador de Portugal. In: IV Congresso Brasileiro de Computação, 2004. **Anais eletrônicos...** Disponível em: < http://www.ufrgs.br/niee/eventos/CBCOMP/2004/html/pdf/Algoritmos/t170100165_3.pdf>. Acesso em 02 junho 2015.

SANTOS, J. C. S; RIBEIRO, A. R. L. JOnline: proposta preliminar de um juiz online didático para o ensino de programação. In: XXII Simpósio Brasileiro de Informática na Educação e XVII Workshop de Informática na Escola, 2011, Acaraju. **Anais eletrônicos...** Acaraju, 2011. Disponível em: <http://www.lbd.dcc.ufmg.br/colecoes/sbie/2011/00119.pdf>. Acesso em: 28 abril 2015.

SCAICO, P. D. et al. Programação no Ensino Médio: Uma abordagem de Ensino Orientado ao *Design* com Scratch. In: Workshop de Informática na Escola, 2012. **Anais eletrônicos...** Disponível em: <http://ceie-sbc.educacao.ws/pub/index.php/wie/article/view/2112>. Acesso em: 17 maio 2015.

SCRATCH. **Crie histórias, jogos e animações:** compartilhe na internet. Disponível em: <http://scratch.mit.edu/>. Acesso em 12 maio 2015.

SOUZA, C. M. de. VisuAlg – Ferramenta de Apoio ao Ensino de Programação. **Revista TECCEN**. USS, Vassouras, V. 2, n. 2, setembro 2009. Disponível em: <http://www.uss.br/pages/revistas/revistateccen/V2N22009/ArtigoVisuAlgSOUZA.pdf>. Acesso em: 17 maio 2015.

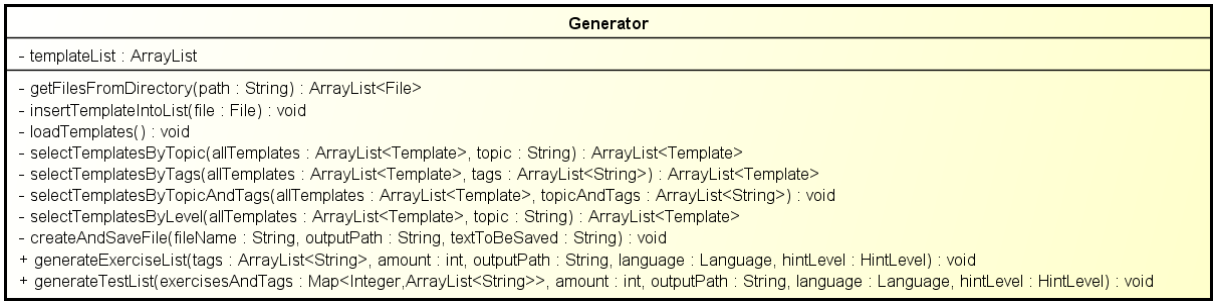
SWACHA, J; BASZURO, P. **Gamification-based e-learning Platform for Computer Programming Education**. In: X World Conference on Computers in Education, 2013. Disponível em: <http://wcce2013.umk.pl/publications/v1/V1.14_125-Swacha-fullR-FPR.pdf>. Acesso em: 09 maio 2015.

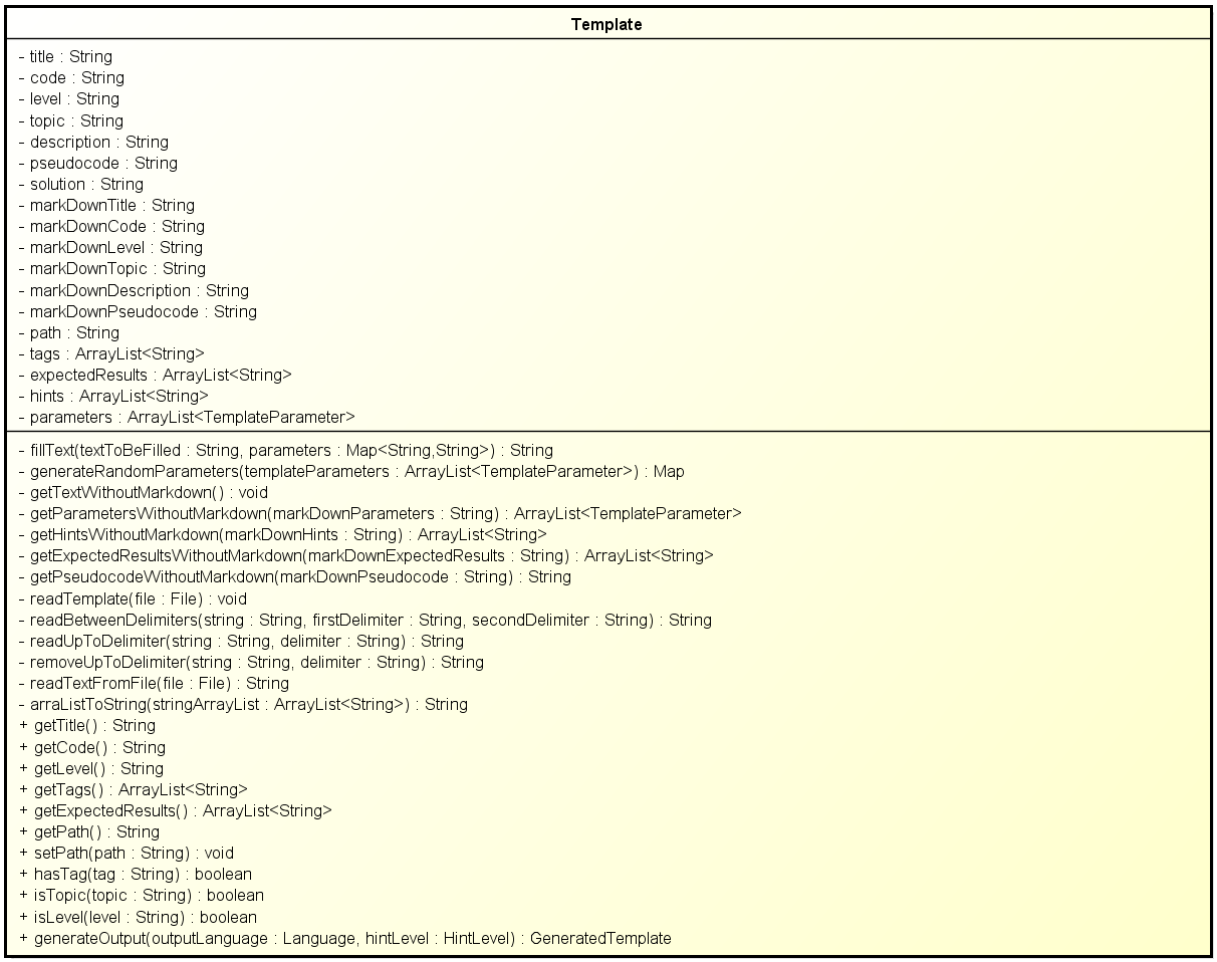
YUAN, L; POWELL, S. **MOOCs and Open Education:** Implications for Higher Education. CETIS, 2013. Disponível em: <http://publications.cetis.ac.uk/wp-content/uploads/2013/03/MOOCs-and-Open-Education.pdf>. Acesso em: 21 abril 2015.

APÊNDICES

APÊNDICE A: DETALHAMENTO DAS CLASSES IMPLEMENTADAS

A seguir serão detalhadas as classes implementadas no sistema gerador de exercícios, exibindo cada método desenvolvido. Entretanto, é importante salientar que apenas as classes criadas na etapa de desenvolvimento do módulo gerador serão exibidas. A classe TranslatorLexer não será detalhada pois foi gerada automaticamente no processo de criação do tradutor. Além disso, apenas foram detalhados os métodos implementados pelo autor na classe TranslatorParser.





FreeMarkerEngine

- cfg : Configuration

+ fillTemplate(templateToBeFilled : String, mapParameters Map<String,String> : int) : String
+ configureFreeMarker( ) : void

GeneratedTemplate

- title : String
- code : String
- level : String
- topic : String
- solution : String
- description : String
- pseudocode : String
- tags : ArrayList<String>
- expectedResults : ArrayList<String>
- hints : ArrayList<String>

+ setTitle(title : String) : void
+ setCode(code : String) : void
+ setLevel(level : String) : void
+ setTopic(topic : String) : void
+ setSolution(solution : String) : void
+ setDescription(description : String) : void
+ setPseudocode(pseudocode : String) : void
+ setTags(tags : ArrayList<String>) : void
+ setHints(hints : ArrayList<String>) : void
+ setExpectedResults(expectedResults : ArrayList<String>) : void
+ getTitle( ) : String
+ getCode( ) : String
+ getLevel( ) : String
+ getTopic( ) : String
+ getSolution( ) : String
+ getDescription( ) : String
+ getPseudocode( ) : String
+ getTags( ) : String
+ getHints( ) : String
+ getExpectedResults( ) : String

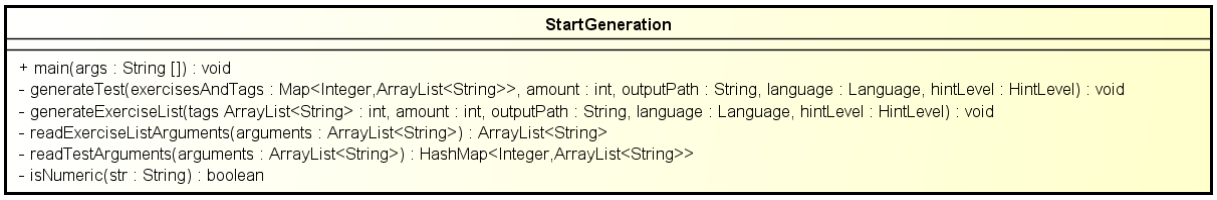
TemplateParameter

- parameterName : String
- parameterType : Type
- integerMinValue : int
- integerMaxValue : int
- doubleMinValue : double
- doubleMaxValue : double
- characterMinValue : char
- characterMaxValue : char
- stringBounds : ArrayList<String>

- splitTemplateParameter(templateParameter : String) : void
- setType(type : String) : void
- setBounds(bounds : String) : void
- setIntegerBounds(bounds : String) : void
- setDoubleBounds(bounds : String) : void
- setCharacterBounds(bounds : String) : void
- setStringBounds(bounds : String) : void
+ getParameterName( ) : String
+ getParameterType( ) : Type
+ getRandomValue( ) : String
- getRandomInteger( ) : String
- getRandomDouble( ) : String
- getRandomCharacter( ) : String
- getRandomString( ) : String
- getRandomBoolean( ) : String

TranslatorParser

+ translate(pseudocode : String) : void
+ setHintLevel(hintLevel : HintLevel) : void
+ setOutputLanguage(language : Language) : void
- clearAll( ) : void
+ getHints( ) : ArrayList<String>
+ getSolution( ) : String


StartGeneration
+ main(args : String []) : void
- generateTest(exercisesAndTags : Map<Integer,ArrayList<String>>, amount : int, outputPath : String, language : Language, hintLevel : HintLevel) : void
- generateExerciseList(tags ArrayList<String> : int, amount : int, outputPath : String, language : Language, hintLevel : HintLevel) : void
- readExerciseListArguments(arguments : ArrayList<String>) : ArrayList<String>
- readTestArguments(arguments : ArrayList<String>) : HashMap<Integer,ArrayList<String>>
- isNumeric(str : String) : boolean